1997-06

Implementing Closed-Loop Control Algorithms for DC-to-DC Converters and ARCP Inverters Using the Universal Controller

Hanson, Ronald J

Ft. Belvoir Defense Technical Information Center

http://hdl.handle.net/10945/8509

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

IMPLEMENTING CLOSED-LOOP CONTROL ALGORITHMS FOR DC-TO-DC CONVERTERS AND ARCP INVERTERS USING THE UNIVERSAL CONTROLLER

by

Ronald J. Hanson

June, 1997

Thesis Advisor: John G. Ciezki

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE IMPLEMENTING CLOSED-LOOP CONTROL ALGORITHMS FOR DC-TO-DC CONVERTERS AND ARCP INVERTERS USING THE UNIVERSAL CONTROLLER	5. FUNDING NUMBERS
6. AUTHOR(S) Ronald J. Hanson	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The objective of this thesis is to investigate the use of the Universal Controller to control the DC-to-DC power converter and the Auxiliary Resonant Commutated Pole (ARCP) power inverter. These power electronic devices are central to the development of a DC Zonal Electric Distribution System (DC ZEDS) that is scheduled for application in the twenty-first century surface combatant (SC-21). The development of appropriate closed-loop controls is a key element to this design process. The Universal Controller is a digital controller that was developed by personnel at the Naval Surface Warfare Center (NSWC), Annapolis, Maryland. The basic operation and control of the DC-to-DC buck converter and the ARCP inverter are described, with emphasis placed on the advantages of DSP control. A complete investigation of the hardware that comprises the controller and how to program the controller to implement closed-loop control is undertaken.

Previous studies have developed control algorithms that have been tested through simulation and analog hardware. In this research endeavor these control algorithms, particularly the one relevant to the DC-to-DC converter, are implemented using the Universal Controller to validate operations. Finally, a flow path for implementing the closed-loop control of the ARCP unit is discussed and recommendations for improvements in future designs are outlined.

14. SUBJECT TERMS dc-to-dc buck converter, auxiliary resonant commutated pole converter, universal controller, dsp control of power converters, texas instruments tms320c30	15. NUMBER OF PAGES 149
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

IMPLEMENTING CLOSED-LOOP CONTROL ALGORITHMS FOR DC-TO-DC CONVERTERS AND ARCP INVERTERS USING THE UNIVERSAL CONTROLLER

Ronald J. Hanson
Lieutenant, United States Navy
B.S., University of North Dakota, 1988

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1997

ABSTRACT

The objective of this thesis is to investigate the use of the Universal Controller to control the DC-to-DC power converter and the Auxiliary Resonant Commutated Pole (ARCP) power inverter. These power electronic devices are central to the development of a DC Zonal Electric Distribution System (DC ZEDS) that is scheduled for application in the twenty-first century surface combatant (SC-21). The development of appropriate closed-loop controls is a key element to this design process. The Universal Controller is a digital controller that was developed by personnel at the Naval Surface Warfare Center (NSWC), Annapolis, Maryland. The basic operation and control of the DC-to-DC buck converter and the ARCP inverter are described, with emphasis placed on the advantages of DSP control. A complete investigation of the hardware that comprises the controller and how to program the controller to implement closed-loop control is undertaken.

Previous studies have developed control algorithms that have been tested through simulation and analog hardware. In this research endeavor these control algorithms, particularly the one relevant to the DC-to-DC converter, are implemented using the Universal Controller to validate operations. Finally, a flow path for implementing the closed-loop control of the ARCP unit is discussed and recommendations for improvements in future designs are outlined.

TABLE OF CONTENTS

I. INTRODUCTION

A. DC ZONAL ELECTRICAL DISTRIBUTION SYSTEM

Improvements in semiconductor technology have sparked a power electronics revolution. In order to best take advantage of these improvements, a new architecture for shipboard power distribution has been proposed called the DC Zonal Electrical Distribution System (DC ZEDS) [Ref. 1]. In this new distribution proposal, the main busses are DC with desired levels being 1000 V and above. The ship is divided into zones, and each zone contains two common energy conversion devices which are fed

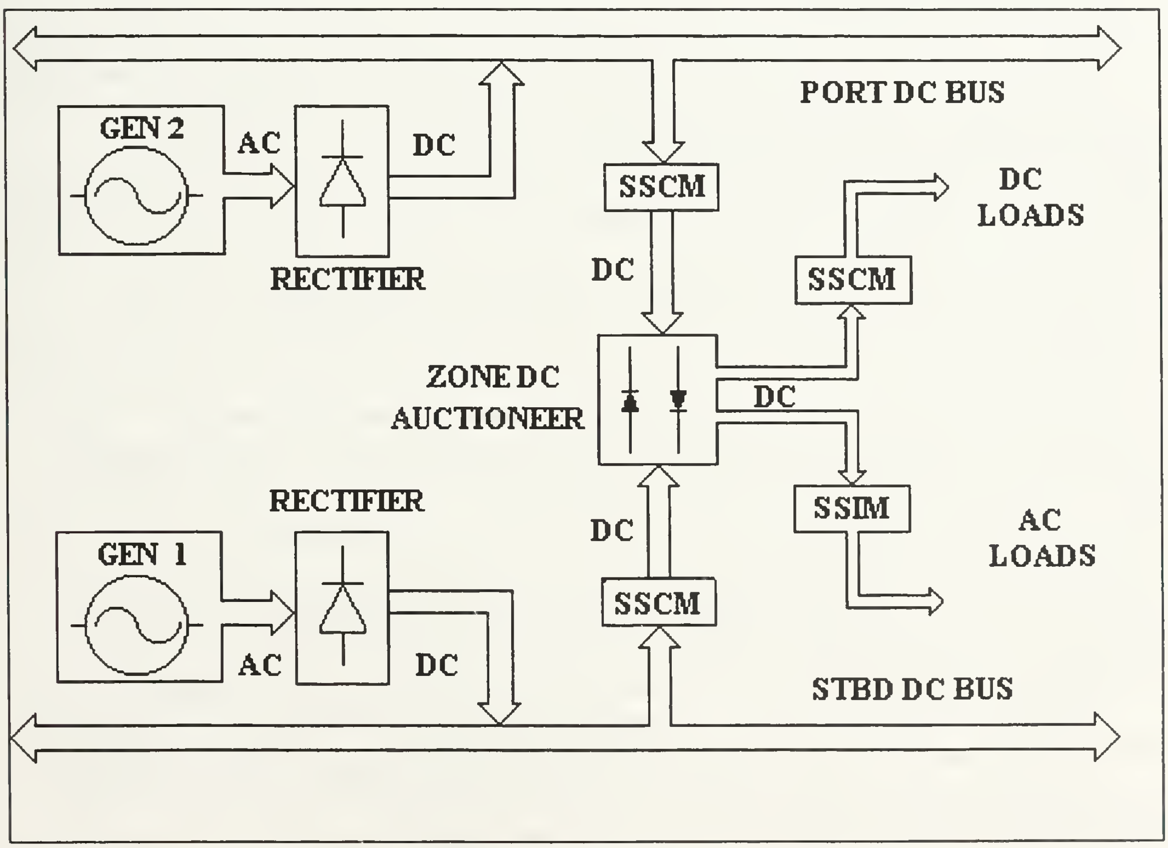

Figure 1-1 - Integrated Power System (from [Ref. 2])

from the DC busses. As shown in Figure 1-1, there are at least two DC busses, a port bus and a starboard bus. The DC is distributed from the source(s) into the separate zones [Ref. 2]. Each zone contains a number of Ship Service Converter Modules (SSCM) and Inverter Modules (SSIM). Each zone has two primary SSCMs which are used to step-down the main DC bus voltage to a regulated level for use in the zone. In this way the SSCM not only acts as a preregulator but also acts as a buffer and implements fault protection for each zone. Electric loads within the zone are fed by either SSCMs or SSIMs depending on the load's requirement for DC or AC power.

B. RESEARCH FOCUS

The focus of this thesis is on controlling the SSCM and SSIM using the Universal Controller. Both the SSCM and SSIM are particular examples of Power Electronic Building Blocks (PEBBs) or devices that integrate power conversion, sensing and control into one package. The Universal Controller, designed by the engineers at the Naval Surface Warfare Center (NSWC) in Annapolis, Maryland, is a digital controller built to demonstrate the Power Electronic Building Blocks (PEBB) software control capability [Ref. 3]. The main advantage of using a digital controller in a program like PEBB is that control algorithm configuration changes can be made in the software and do not need to be re-engineered into the hardware. For example, to change the switching frequency of the SSCM that uses an analog controller one must physically change various components on the control board. To affect the same change with a digital controller, one must only implement a change in the software.

The Universal Controller was built by the Naval Postgraduate School (NPS) Power Research Group to be used in its research, specifically for the PEBB Network Simulation Testbed [Ref. 4]. The testbed is currently being developed to help garner understanding on how all of the energy conversion devices within a distribution zone interact. The Universal controller will enable the researchers to test different control algorithms by changing only the software and not the hardware. In this manner, different control bandwidths can be implemented so that negative-impedance effects can be readily investigated.

The Universal controller has no manual and therefore the focus of this thesis is on documenting how the controller works and describing how to implement desired control algorithms for the SSCM and the SSIM. In Chapter II the basic operation of the SSCM and SSIM are presented and the equations governing their circuit behavior are discussed. Next, Chapter III deals with the digital signal processing (DSP) hardware contained on boards available at NPS and then discusses the hardware associated with the Universal Controller. The hardware must be controlled by a software program, and in Chapter IV the software provided by NSWC is investigated. The software provided by NSWC is used to initialize the control board and to allow for a PC interface with the board. This allows data to be downloaded to the controller via the keyboard on the host PC. Next the DSP control implementation is presented in Chapter V. The final chapter contains a summary of the research work, notable conclusions and recommendations for future work.

II. POWER CONVERTERS

A. INTRODUCTION

The primary power electronic elements in the DC zonal distribution system are the Ship Service Converter Modules (SSCM) and the Ship Service Inverter Modules (SSIM). The SSCM processes the high-voltage primary DC bus power and produces regulated power at a lower DC voltage. The SSIM takes DC power and converts it to single-phase or three-phase AC power. The topology used for the SSCM is the DC-to-DC buck chopper. The topology for the SSIM is the auxiliary resonant commutated pole (ARCP) inverter. Both the buck chopper and the ARCP utilize power semiconductor devices controlled by signal electronics to perform the power conversion. This chapter contains a brief description of the operation and control of the buck chopper and the ARCP.

B. DC-TO-DC CONVERTER

The buck chopper, or step-down converter, is a DC-to-DC converter in which the output voltage is always less than the input voltage. The topology for the buck chopper is shown in Figure (2-1). By controlling the amount of time that the switch is closed for a given switching period, a "chopped" voltage waveform between zero and the DC input voltage is realized across the circuit diode. The LC filter then acts to filter out the high-frequency harmonics and extract the average value. Closed-loop control is almost always required to guarantee acceptable transient and steady-state response.

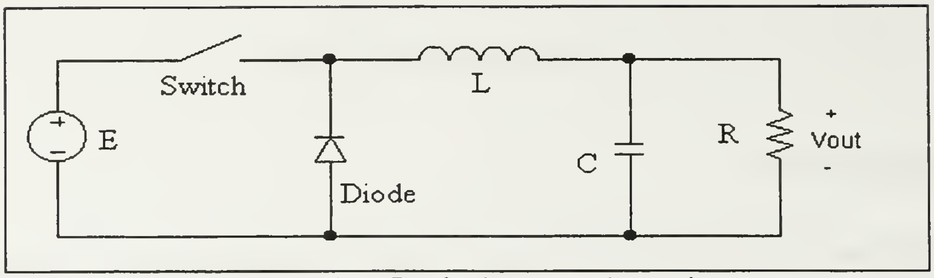

Figure 2-1, Buck chopper schematic

1. Basic Buck Chopper Operation

The majority of the material presented on the buck chopper is derived from Mohan [Ref. 5]. The voltage output of the buck chopper depends on the duty cycle, which is the ratio of the time the switch is "on" to the switching period, and the input voltage. During continuous current operation, the following steady-state input/output relationship holds.

$$D = \frac{t_{on}}{T_s} \tag{2-1}$$

$$V_{out} = DE \tag{2-2}$$

Assuming that C is sufficiently large, V_{out} will remain constant. Figure (2-2) illustrates typical relationships between the input voltage, duty cycle and output voltage.

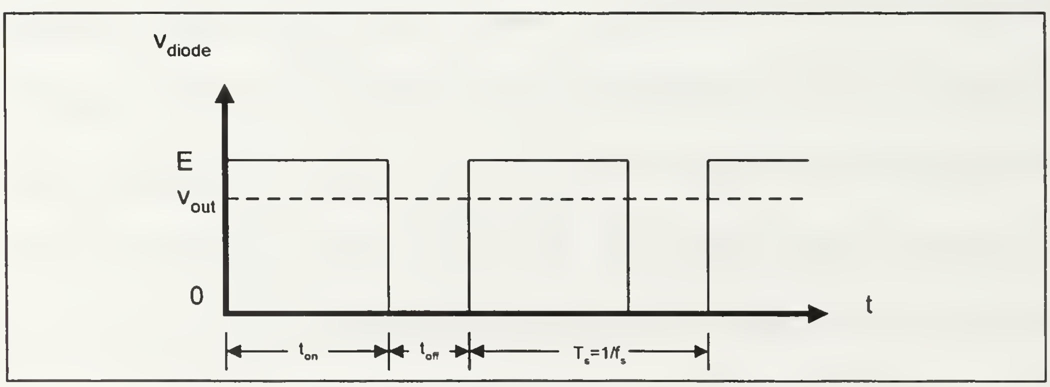

Figure 2-2, Buck chopper voltage waveforms

There are two modes of operation for the buck chopper, continuous and discontinuous. For the continuous case, the inductor current does not go to zero during the switching period, whereas in the discontinuous mode it does. Figure (2-3) shows the inductor current for both cases. It is desirable to operate with continuous current since in this mode the buck chopper is equivalent to a DC transformer with the turns ratio controlled electronically via the duty cycle. The energy storage capacity of the inductor determines the mode of operation. The value of the inductance at the boundary between continuous and discontinuous conduction modes is called the critical inductance and is given by Equation (2-3).

$$L_{crit} = \frac{T_s R}{2}(1 - D) \qquad (2\text{-}3)$$

where R is the attached load resistance.

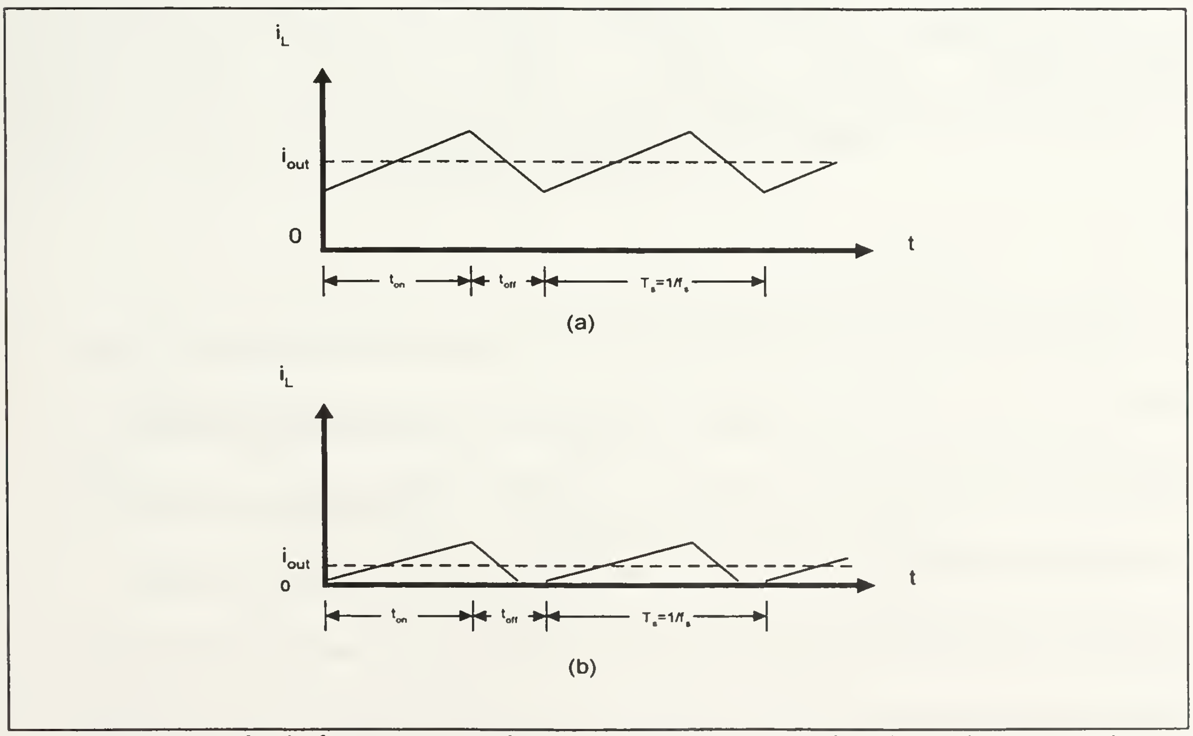

Figure 2-3, The inductor current for (a) continuous mode; (b) discontinuous mode

2. Specifications

A medium-power DC zonal electric distribution network simulation testbed is being fabricated at the Naval Postgraduate School. The testbed is intended to include converters with device ratings and specifications that will provide good representative data sets. These data sets may then be used to assess stability boundaries, transient response and nonlinear interaction between converters and converter controllers. The buck chopper must meet the specifications shown in Table (2-1). The rated full load resistance is 10 Ω while the minimum load before transitioning into discontinuous current operation is 100 Ω. As dictated from the specifications, the nominal duty cycle is 0.693.

Rated Power	Switching Frequency	Input Voltage	Output Voltage	Input Current	Output Current	Mode of Operation
3 kW	20 kHz	300 V	208 V	10 A	14.5 A	Continuous

Table 2-1, Buck chopper specifications

3. Components

The components shown in Figure (2-1) make up the basic buck chopper circuit. The switch is an Insulated Gate Bipolar Transistor (IGBT) made by International Rectifier [Ref. 6]. The maximum voltage V_{CE} is 600 V and the maximum current I_C is 90 A. The switch can be hard switched at a frequency of 25 kHz. The IGBT package contains two freewheeling diodes. Both diodes are used, one across the IGBT to handle reverse currents through the switch, and the other as shown in Figure (2-1). The values of

the other components are listed in Table (2-2). The L_{in} and C_{in} components form an input filter to the basic buck structure depicted in Figure (2-1). If a rectifier is used to supply the input DC, this filter operates to attenuate the resultant harmonics.

L_{in}	C_{in}	L	C
425 μH	2000 μF	1100 μH	2000 μF

Table 2-2, Component values

4. Interface

The IGBT requires a positive V_{GE} to turn "on" and a negative V_{GE} to turn "off". The driver circuit [Ref. 7] optically isolates the control signal from the IGBT by use of a TOSHIBA TLP 250 gate drive photo IC coupler. This driver amplifies the turn-on and turn-off voltages and protects the controller from the large voltage swings occurring across the IGBT. Figure (2-4) illustrates a simplified gate driver for the IGBT. The switch is on when the control signal (light) is "on".

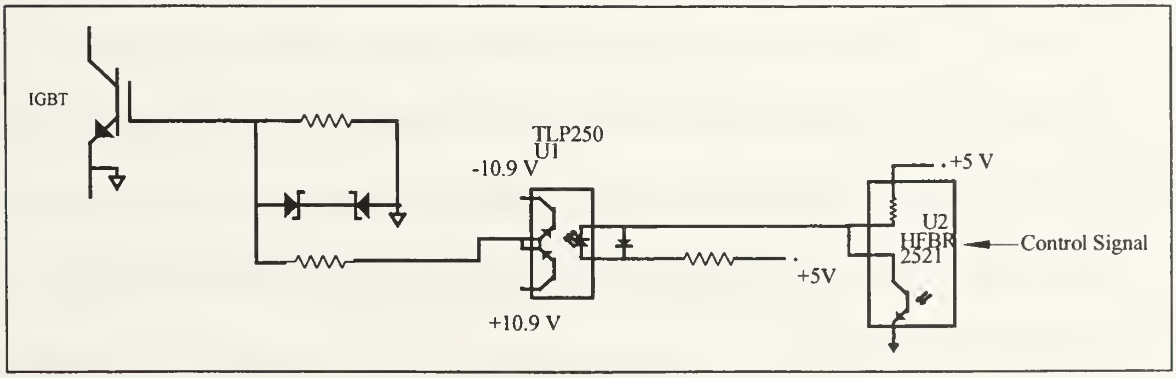

Figure 2-4, Simplified IGBT driver circuit.

5. Control

The buck chopper must be stable, allow for fast transient response, and allow parallel units to share power equally. The control algorithm [Ref. 7] to be implemented is:

$$d(t) = Dss - \left(h_v + h_n \int dt\right)\left(v_0 - v_{ref} - \frac{i_0}{10}\right) - h_i\left(i_L - i_0\right) \tag{2-4}$$

where,

d(t) = time-varying duty cycle	Dss = steady-state duty cycle
h_v = voltage gain	h_n = integrator gain
v_0 = buck output voltage	h_i = current gain
v_{ref} = reference voltage	i_0 = load current
i_L = inductor current	

The quantity (v_o - v_{ref} - i_o/10) represents the error between the actual buck output voltage and what is desired. The i_o/10 term represents a voltage droop which is required for paralleling two units. The control law uses a current gain to force the inductor current to track changes in the output current. In addition, the proportional voltage error signal is there for stability while the integral term guarantees zero steady-state error. The steady-state duty cycle term represents feed-forward action that compensates for changes in the input voltage. The controller must also be able to handle auxiliary functions such as limiting the current, temperature sensing and buck chopper startup. Additional details may be found in reference [7].

6. Motivation for DSP

The control algorithm described above must be implemented and tested. Designing the control using software and a universal DSP controller will significantly enhance the testing and evaluation of the DC-to-DC converter by allowing flexibility and speed in changing control algorithms, adjusting gains, and investigating different switching frequencies. Such modifications using the analog controller described in reference [7] are impractical and generally involve significant efforts in unsoldering components and inserting and tuning new ones. The PEBB technology [Ref. 3] is based on software control capability and, in particular, configuring a single controller to control multiple applications such as both the buck chopper and the ARCP inverter.

C. DC-TO-AC INVERTER

The switch-mode DC-to-AC inverter is a power conversion device which takes DC power and produces an approximation to a sinusoidal AC output where both the frequency and amplitude of the output can be controlled.

1. Basic Operation

The SSIM is a three-phase inverter. To help explain the basic operation of the switch-mode inverter only one phase of the inverter will be discussed. Switch-mode inverters operate by electronically controlling the "on" times of the switches. The switch can be hard-switched, which results in high switching stresses and power loss, or soft-switched (ARCP) which reduces the switching stresses and power loss.

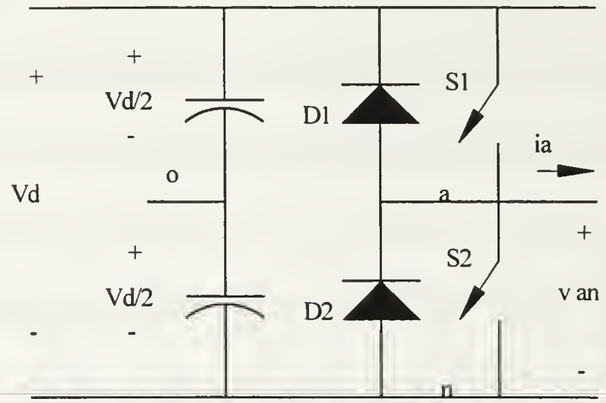

Figure 2-5, One phase hard-switched inverter

a) Hard-Switched 3-Phase Inverter

The simplified schematic of one phase of a hard-switched inverter is shown in Figure (2-5). For this circuit to operate, the switches S1 and S2 open and close in a predetermined fashion to produce the AC output. In steady state for any given instant, only one switch will be closed at a time and either the diode or switch will be carrying the current depending on the direction of the current flow. For example when S1 is gated and S2 is open, $v_{an} = V_d$. If the current flow is out, $i_a > 0$, S1 is conducting. If the current flow is in, $i_a < 0$, D1 is conducting. When S2 is gated and S1 is open, $v_{an} = 0$. If $i_a > 0$, D2 conducts. If $i_a < 0$, S2 conducts. By connecting the return path of the load to terminal o, the output v_{ao} alternates between $\pm V_{d/2}$. The remaining two phases operate in an analogous manner but have their gating signals displaced electrically by 120°.

b) Auxiliary Resonant Commutated Pole Inverter

The ARCP inverter allows for zero-switching voltages across the IGBTs which reduces the power loss and switching stresses associated with the hard-switched converter. The specific operation of an ARCP can be found in [Ref. 8]. Figure (2-6) shows one phase of the ARCP. The auxiliary circuit switches A1 and A2 are controlled by sensing the direction of the output current and the voltages across the switches so that the resonant current, i_r, will either pump current to the circuit ($i_r > 0$) when D1 or D2 is conducting, or sink current from the circuit ($i_r < 0$) when i_o is below a certain threshold and conducting through S1 or S2. In particular, these circuit switches are only gated during S1 and S2 switching transitions and the high-frequency resonant current pulse acts to supply the needed energy to achieve zero-voltage switching.

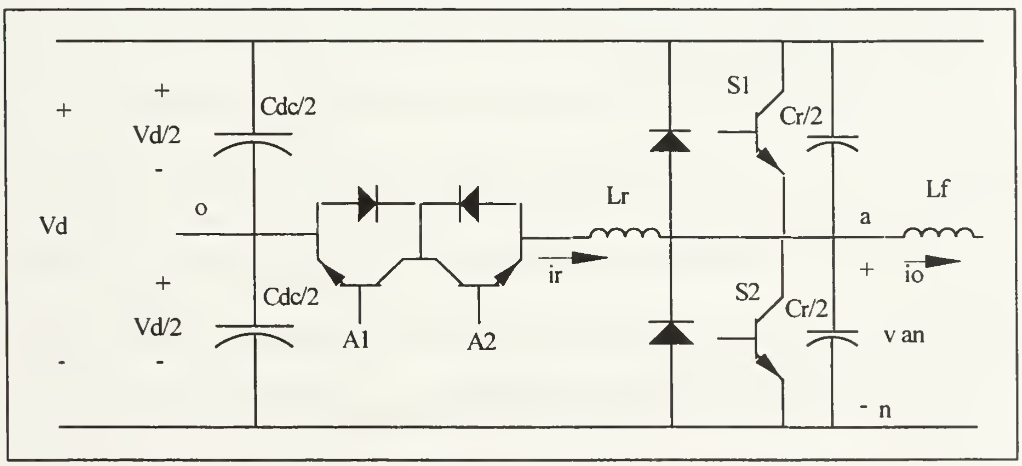

Figure 2-6, Single phase of the ARCP

2. Specifications

The ARCP was designed and assembled by the Applied Research Laboratory, Penn State University [Ref. 9] with the following specifications listed in Table (2-3).

Operating Specifications		
	PWM operating frequency	0 to 50kHz
	Minimum dead time	1 μs
	Nominal resonant frequency	250 kHz
	Maximum load rate (di/dt)	36 A/μs
Input/Output Specifications		
	Rated dc bus voltage	400 V
	Maximum overvoltage	150 V
	Output voltage (3ϕ-ac)	220 V (rms,line to line)
	Output current (3ϕ-ac)	17.7 A (rms)
	Output voltage (dc)	400 V
	Output current (dc)	25 A
	Power supply voltage	24 V ± 2 V
	Maximum power supply current @ 24 V	2.2 A
	Drive signal inputs	fiber optic
Current Feedback		
	Isolation from main power bridge	< 10 MΩ
	Reference	user ground
	Bandwidth	200 kHz
	Scaling (V/A) - This is the scaling of the sensed current value (1V=10 A)	0.1 V/A
	Limit - This is the maximum voltage allowed by the current sensor.	± 2.75 V

Table 2-3, The ARCP specifications

The units were designed to contain three separate pole (phase) boards to facilitate operation as a three-phase unit or three separate single phases. A control board was also

incorporated to generate the actual main and auxiliary switch gating signals given the monitored variables and the commanded main switch signals.

3. Components

Due to the high resonant frequency (250 kHz) of the auxiliary circuit, switches A1 and A2 on Figure (2-6) were chosen to be APT4520 MOSFETs. These devices have a voltage rating of 450 V and an on-state resistance of 0.20Ω. The pulse-by-pulse peak current that maybe conducted through the mosFETs is 92 A for a maximum period of 10 μs. Utilization of these devices allowed resonant cycles with a frequency as high as 325 kHz.

The main switches S1 and S2 were chosen to be IRGPC50U IGBTs. These devices allow peak current carrying capabilities of 50 A at a switching frequency of 10 kHz. Since the unit is designed to accommodate switching frequencies up to 50 kHz, correct ARCP operation (soft switching) is essential above 10kHz.

The ARCP requires two sets of diodes, one set for the main switches and the other set for the auxiliary circuit. The diodes chosen for the main switches are HEXFRED HFA30PA60C diodes, which offered reasonable conduction losses and short recovery times. The auxiliary diodes primary requirement is speed of recovery to prevent circulating currents among the switches. The devices chosen were the MUR3040, which allow a 50 ns reverse recovery time.

The passive component values are as follows: $L_r \approx 2.8\ \mu H$, $C_{dc}/2 \approx 10\ \mu F$, and $C_r/2 \approx .05\ \mu F$. The inductance L_f listed in Figure (2-6) is a part of the attached load and is required to satisfy the di/dt specifications of the converter.

4. Interface

The ARCP inverter requires the control of four switches for each phase of the inverter. The auxiliary switches are controlled internally by the SSIM unit while the gating signals for the main switches are specified by an external controller. Figure (2-7) shows the circuit block diagram for a single phase of the ARCP. The complete ARCP contains three separate phase boards each with an identical block diagram. From this diagram, it is clear that each phase is composed of the following sub-circuits: two main drive and voltage sense circuits, two inner controller and auxiliary switch drive circuits, and an outer controller [Ref. 9].

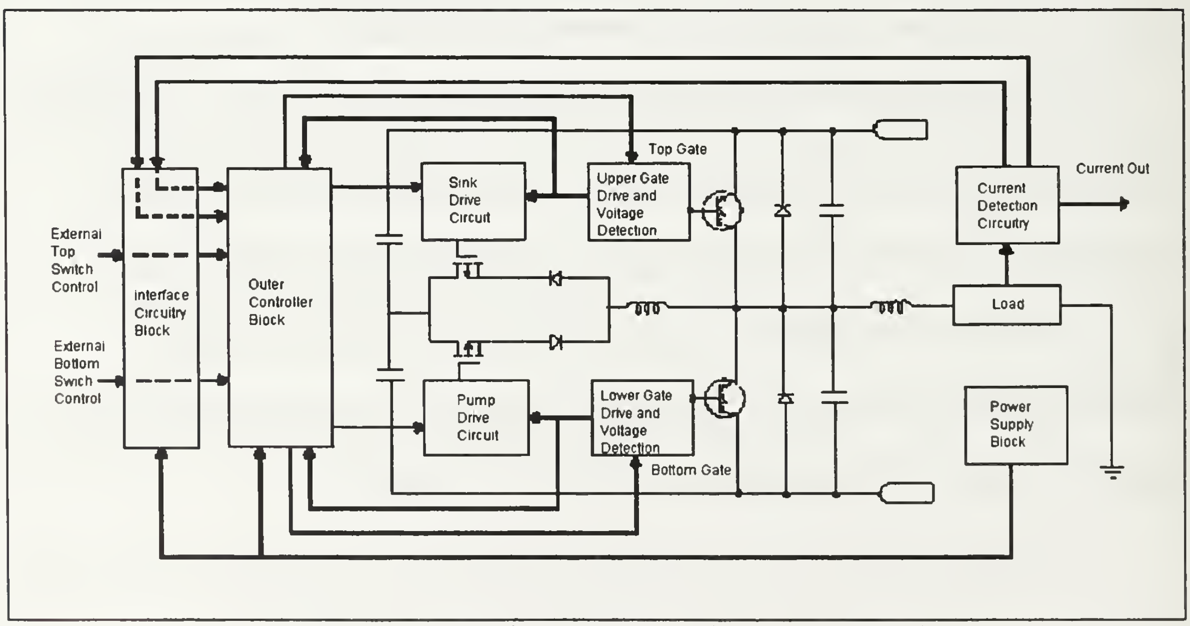

Figure 2-7, Circuit block diagram of the ARCP [Ref. 9]

The main drive circuitry takes the external control signal (light) for the main switches and converts the signal into the proper turn-on or turn-off gate voltages. This circuit also detects the voltage across the switch which then can be monitored by the user.

The inner controller, labeled pump drive circuit and sink drive circuit on Figure (2-7), controls the auxiliary switches. The pump drive circuit turns “on” the bottom auxiliary switch so that current is added to the system, whereas the sink drive circuit turns “on” the top auxiliary switch so that it acts as a current sink.

The outer controller block controls the timing of the resonant switches. This is accomplished by determining the direction and magnitude of the output current, via the current detection circuitry, and the voltage across the main switches. The outer controller also feeds the user control signals to the main switch drivers.

The current detection circuitry and power supply block are not physically located on the phase board but the signals generated by these circuits and the user control are connected to the outer controller via the interface circuitry block.

5. Control

In order to produce a sinusoidal output with the amplitude and frequency controllable, a more complex switching method then the buck chopper is used. The method is called sine-triangle pulse width modulation (PWM) [Ref. 5]. Figure (2-7) shows a triangular waveform superimposed on a sinusoidal waveform. The triangular waveform is at switching frequency (f_s) which establishes the frequency that the inverter switches are gated. The sinusoidal waveform is the control signal ($v_{control}$) and is used to

modulate the switch duty ratio and its frequency (f_1) is the desired fundamental frequency of the inverter output. The switches in the ARCP inverter of Figure (2-6) are controlled based on the comparison of $v_{control}$ and v_{tri}. When $v_{control} > v_{tri}$ S1 is "on" and $v_{ao} = V_d/2$; when $v_{control} < v_{tri}$ S2 is "on" and $v_{ao} = -V_d/2$. The amplitude is controlled by the following ratio:

$$m_a = \frac{\hat{V}_{control}}{\hat{V}_{tri}} \tag{2-5}$$

While in the linear range of control, the peak amplitude of the fundamental frequency component is

$$\hat{V}_{a01} = m_a \frac{V_d}{2} \tag{2-6}$$

Closed-loop control of an inverter generally seeks to regulate either the three-phase voltages or the three-phase currents out of the inverter. This is accomplished by externally generating reference signals and comparing these with the actual measured quantities. The resultant error signals are then used to adjust the modulating waveform depicted in Figure (2-8). Several closed-loop inverter control schemes were investigated and reported on in [Ref. 10].

6. Motivation for DSP

The ARCP must be tested with different load configurations and control laws. The use of a DSP controller will allow flexibility in that the control can be modified in software saving time and resources. The noise inherent in an analog controller is eliminated and this allows for a much cleaner output from the ARCP.

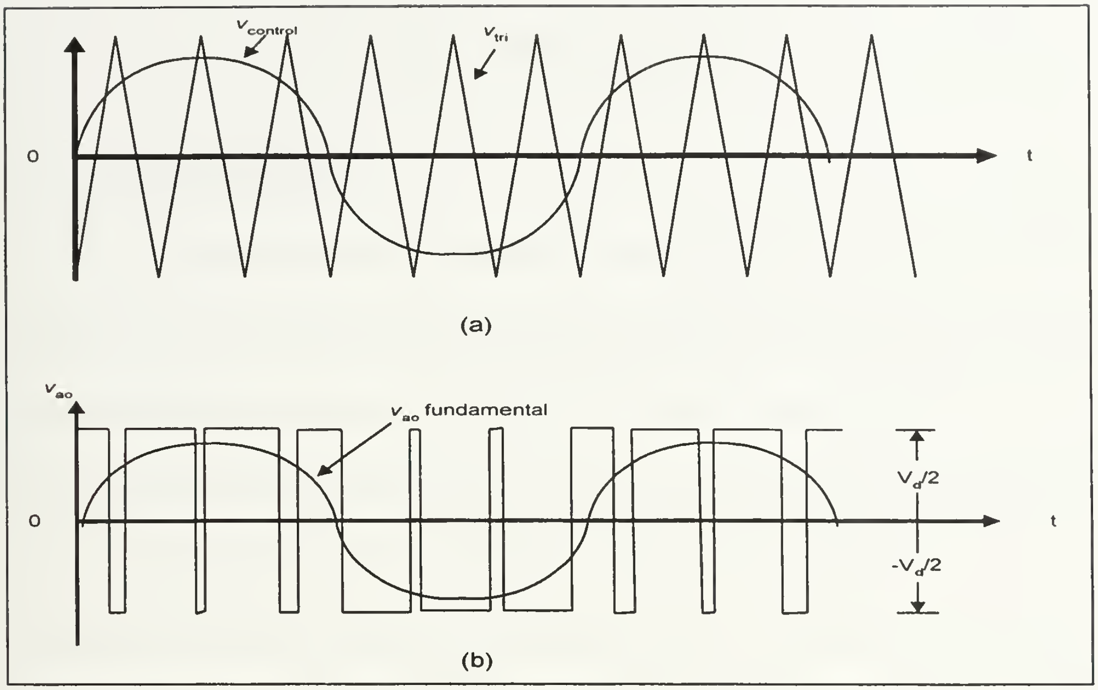

Figure 2-8, PWM (a) the sine-triangle and (b) output waveforms.

It is clear that DSP control is a desired feature for implementing the control of the buck chopper and the ARCP inverter. Personnel at the Naval Surface Warfare Center, Annapolis have developed a control board for the PEBB program called the PEBB Universal Controller. The Universal Controller is currently in the development stage. This thesis will concentrate on using the Universal Controller hardware to implement the control for the buck choppers and the ARCP inverters within the NPS testbed. The Universal Controller's hardware is examined in the next chapter.

III. DSP HARDWARE

A. INTRODUCTION

As discussed in the last chapter, DSP control of the ship service inverter module (SSIM) and the ship service converter module (SSCM) offers flexibility in the design and implementation of various control algorithms. Software control of power electronic converters is certainly not a novel idea. Various successful attempts at controlling devices have been undertaken at NPS [Ref. 10 and 11], but identifying the most appropriate DSP hardware configuration for this type of control has been a problem. One of the unique requirements for controlling either the ARCP inverter or the buck chopper is that the algorithms for each require many signals to be monitored and processed. The major limitation of the commercial boards that have been used in past efforts is the lack of signal inputs available for processing. This problem has motivated engineers at the Naval Surface Warfare Center (NSWC), Annapolis, MD, to design a unique DSP product called the PEBB Universal Controller. The purpose of this research is to implement and validate control algorithms using the NSWC Universal Controller. In this chapter, a general hardware description of this controller and a brief overview of three commercial DSP boards are presented. Aspects of the software control of the Universal Controller are then described in the next chapter.

B. COMMERCIAL DSP BOARDS

The heart of any DSP board is the processor. DSP processors can be divided into two broad categories: general purpose and special purpose [Ref. 12]. Special purpose processors execute specific algorithms such as digital filters, FFTs, and cosine transforms for use in image processing. General purpose microprocessors are basically high-speed microprocessors with hardware architectures and instruction sets tailored for real-time DSP operations. DSP processors make use of the Harvard architecture which employs a separate bus for addresses and data, and different memory locations for data and instructions. This allows for full overlap of instruction fetch and execution. For example, since the program instructions and data lie in separate memory locations, the fetching of the next instruction can overlap the execution of the current instruction. Texas Instruments produces the TMS320 family of general purpose DSP chips. The TMS320 processors use a modified Harvard architecture. In the modified architecture, separate program and data memory spaces are still maintained, but communication between the two memory spaces is permissible. All of the DSP boards that have been encountered in this research effort and are available at NPS contain the TMS320C30/31 DSP processor. These include:

- C30 system board by LSI (SPECTRUM Signal Processing Inc.)
- SBC31 system board by Innovative Integrations Digital Signal Processor
- LD31 system board by dSPACE (digital signal processing and control engineering).

The C30 board features the Texas Instruments TM320C30 general purpose DSP processor. This board contains two banks of off chip memory for a total of 128K words. The on board memory allows for quick access to data. Dual channel 16 bit A/D and D/A converters are included on the board with sampling rates of up to 200 kHz. The board is suitable only for PC/AT and compatibles, since it uses the full 16-bit interface. All communication with the board is via the I/O space of the PC. Access to memory passes are through the dual porting hardware on the board. Reference [13] has the complete details for this board.

The SBC31 board is based on the Texas Instruments TM320C31. The SBC31 is a stand alone card with 128K of on-board Programmable Read Only Memory (PROM) and 32K of Static Random Access Memory (SRAM) for user programs. The PROM contains the program instructions and the SRAM is for data storage and retrieval. The board includes one dual channel 16-bit, 200 kHz A/D converter and 2 dual channel D/A converters. The board contains two serial ports. A remote PC can be connected via the RS232 connector to communicate with the board. This board was used for implementing a digital control algorithm for a DC-to-DC buck chopper previously, but had some serious limitations [Ref. 11]. The most significant limitation was the delay times associated with the A/D converters. The large delay time occurred because of two reasons:

1. One A/D converter was used to sample 4 signals .
2. The A/D converter is double buffered; therefore, two conversion cycles are required before the sampled data is available for computations.

The third DSP board available at NPS is the digital Signal Processing And Control Engineering (dSPACE) DS1102 controller board. This board can be inserted into any 16-bit slot on a PC/AT computer. The DS1102 is based on the Texas Instrument TMS320C31. This board contains two 16-bit A/D converters and two 12-bit A/D converters [Ref. 14]. One of the benefits of the DS1102 is that it comes with software that allows the user to interface with the MathWorks development software MATLAB and SIMULINK. This facilitates the design and implementation of a controller using the graphical modeling capabilities of SIMULINK. C code can then be generated from the SIMULINK design and compiled by the TMS C compiler. This makes the dSPACE board very easy to work with. The board does have significant limitations. As documented in Reference [10], the DS1102 was limited to single-phase operation of the ARCP inverter because of the limited number of I/O ports. In addition, there are conservative bounds on the complexity of the algorithm that maybe loaded form SIMULINK into the board.

As mentioned at the beginning of the chapter NSWC, in conjunction with the PEBB program, developed their own DSP controller. This controller was designed primarily for power electronic circuits, and, specifically, for the intensive input/output requirements of the ARCP.

C. NSWC PEBB UNIVERSAL CONTROLLER

The PEBB Universal Controller was designed to control various applications as shown in Figure (3-1). Presently the application which requires the most I/O support is

the ARCP, which requires as many as 12 gate signals and the ability to convert 10 analog signals to digital signals. Each phase of the ARCP inverter has 4 gates to be controlled:

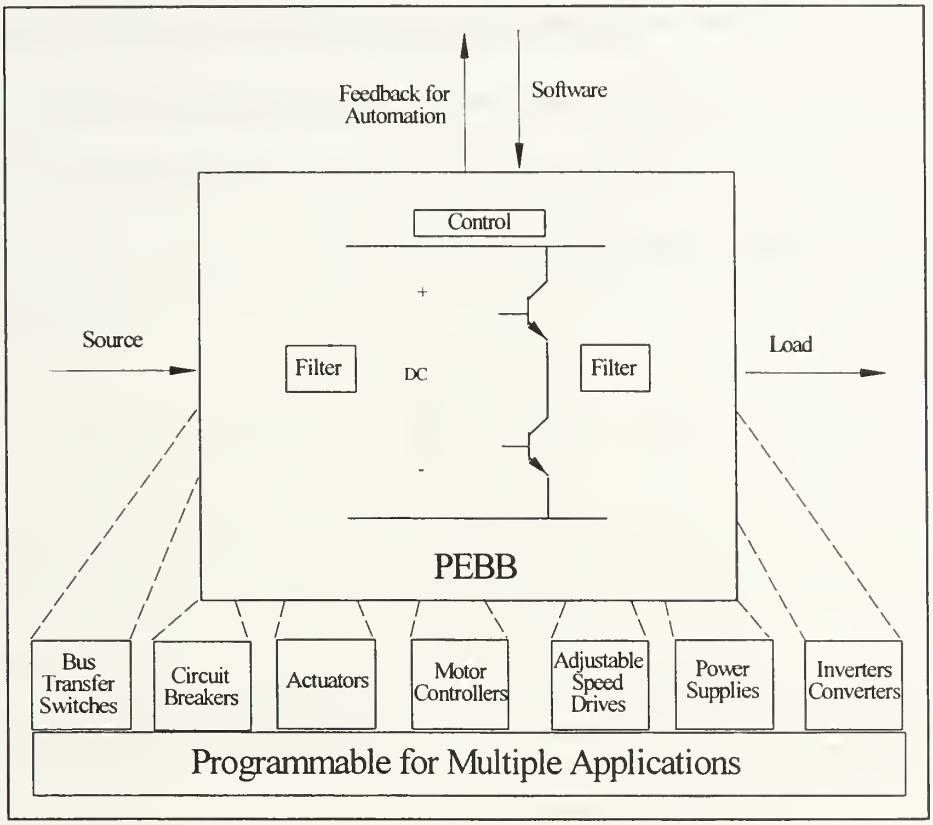

Figure 3-1, PEBB package diagram [Ref. 4]

two main switches and two auxiliary switches (see Chapter II). The units delivered to NPS have a provision wherein the auxiliary switches may be controlled by the unit itself. In general, however, control of all 12 switches requires sampling the output voltage and current of each phase and the input voltage and current. By designing the DSP board to handle the most I/O intensive application, switching to other applications only involves changing the software and not the hardware.

1. General Description

The Universal Controller is built around the TMS320C30 DSP chip and is contained on two boards, the mother board (CPU board) and the daughter board (I/0 board). Figure (3-2) shows the CPU board and I/0 board external connections.

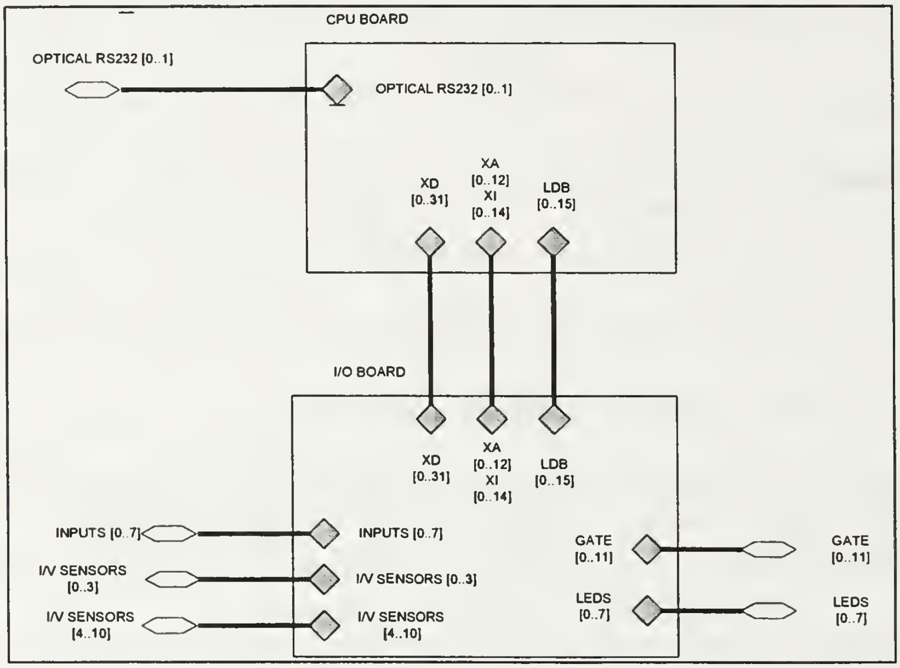

Figure 3-2, Universal controller CPU and I/0 boards [Ref. 15]

Communication with the CPU board is accomplished via the RS232 serial port on a host PC. The signal from the host PC is converted to an optical signal and read by the CPU board. The board also can send information to the PC such as voltage and current values for display on the screen. The I/0 board contains all the voltage and current sensor inputs and also sends the gate signals to the switches. In Figure (3-2), XD and XA correspond to the 32-bit TMS320 expansion data bus and the expansion address bus respectively. The LDB is a 16-bit data bus from the mirocontroller contained on the CPU board. The L signifies that the microcontroller is connected to the left side of the dual port memory

which is discussed below. The I/O board shows the 10 current/voltage sensors (on the left of the diagram) which are connected to the A/D converters and 12 gate connections (on the right of the diagram) which are the optical transmitters used to drive the switching circuits. The LED connection is used to display the current mode of operation (e.g. ARCP control or buck control). The inputs [0..7] (on the left of the diagram) can be used to start and stop the controller locally. The input and LED connections are not used in this research effort.

2. Primary Components

The primary components of the Universal Controller can be divided up into the following five categories: processor, memory, PC interface, analog interface, and timer/counters. A brief description of each of these components follows. A complete listing of all the parts contained in the Universal Controller is given in Appendix A.

a) Processor

The processor is the 181-pin grid array TMS320C30 from Texas Instruments [Ref. 16]. The C30 is operating from a 40 MHz clock which results in a 50 ns instruction cycle time and the performance of 20 million instructions per second. The peak arithmetic performance is 40 million floating-point computations per second when the floating-point multiplier and adder are used in parallel. The C30 contains one 4K x 32-bit, single-cycle, dual-access, on-chip ROM block and two 1K x 32-bit RAM blocks. The C30 can be set up in two different modes, microprocessor mode or microcomputer

mode. The memory map (see Figure 3-3) depends on which mode the processor is running in. The default mode for the Universal Controller is the microprocessor mode.

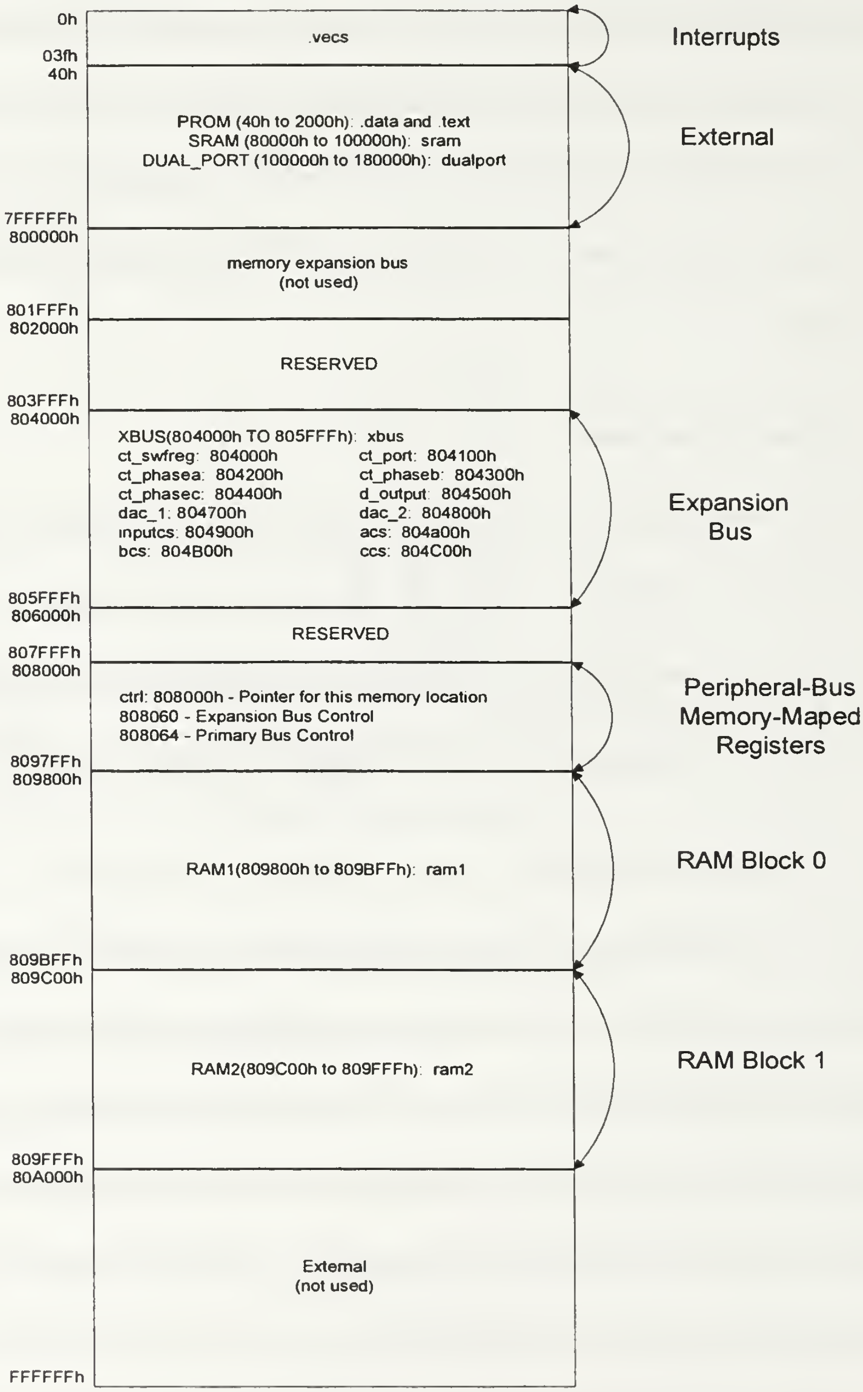

Figure 3-3, Memory map for microprocessor mode.

The details of the memory map will be discussed in Chapter IV. Other key features of the microprocessor are:

- 64 x 32 instruction cache
- 32-bit instruction and data words
- 24-bit addresses
- 40-/32-bit floating-point/integer multiplier and ALU
- Eight extended-precision registers (accumulators)
- Two address generators with eight auxiliary registers and two auxiliary register arithmetic units
- On-chip DMA controller for concurrent I/O and CPU operation
- Parallel ALU and multiplier instructions in a single cycle
- Two 32-bit data buses (24- and 13-bit address)
- Two 32-bit timers
- Two general-purpose external flags; four external interrupts.

Figure (3-4) shows a functional block diagram to illustrate the interrelationships between the various C30 key components. Figure (3-4) does not show the port control registers. The port control registers consist of two 32-bit registers, one for the primary-bus control and the other for the expansion-bus control. The primary bus control is set up to allow for 1 wait state and 1M bank compare to allow for reading from the PROM and static random access memory (SRAM). The expansion bus is set up to allow for 2 wait states which are required for reading the values from the slower A/D converters and writing control words and count values to the counter/timers.

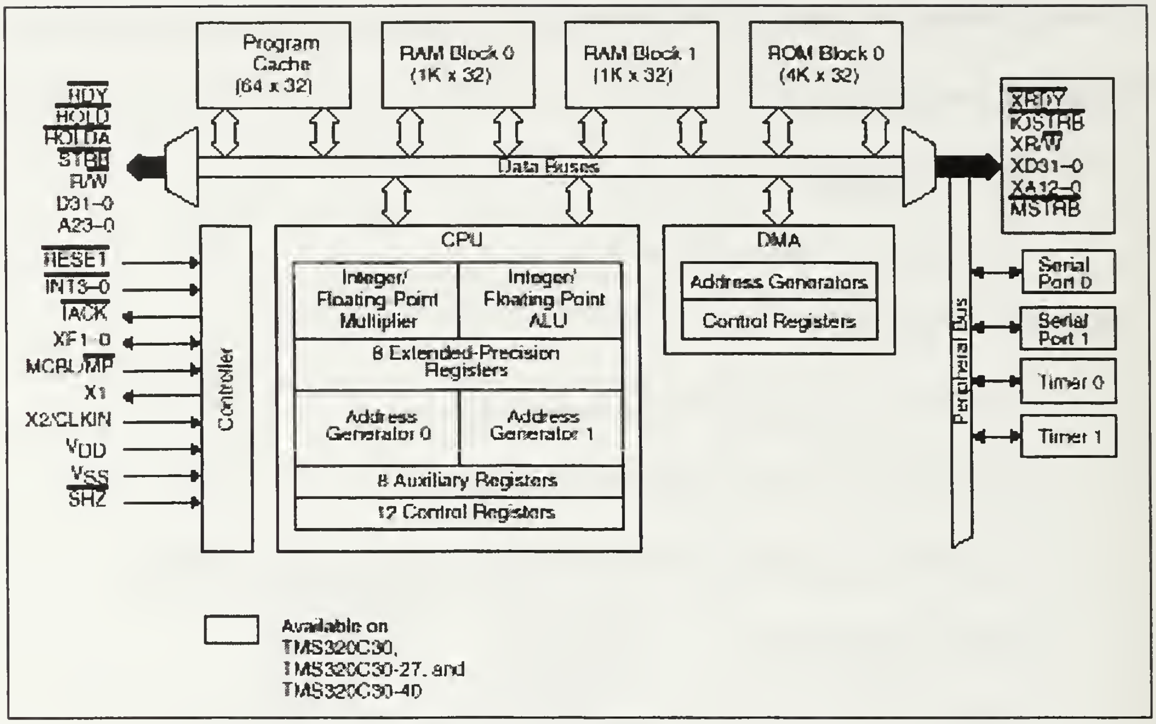

Figure 3-4, The TMS320C3x block diagram [Ref. 16].

b) Memory

The external memory is divided into three memory areas. Figure (3-4) shows the block diagram of the CPU board and the locations for the three memory areas. The first area is the programmable read only memory (PROM). This is where the program code, which is discussed in the next chapter, is located. Four WSI WS57C256F-35 32K x 8 power switched and reprogrammable PROM chips were selected. The TMS320C30's 32-bit primary bus is divided up between the four PROMs, which have 8-bit word memory. An address decoder is used so that a memory read will read the same memory location on all four PROMs simultaneously to generate the 32-bit word for the microprocessor. The CPU board has 28-pin sockets to allow the PROM to be removed

for programming. The PROM is memory mapped to 40h. This means that the address for the top memory location of the PROM is located at 40 hex.

The next memory area is the static RAM. The static RAM is made up of four IDT71256SA fast 32K x 8 CMOS chips. A memory read or write works just like a memory read for the PROM. The static RAM is memory mapped to 80000h. Figure (3-5) shows the location of the static RAM on the CPU board.

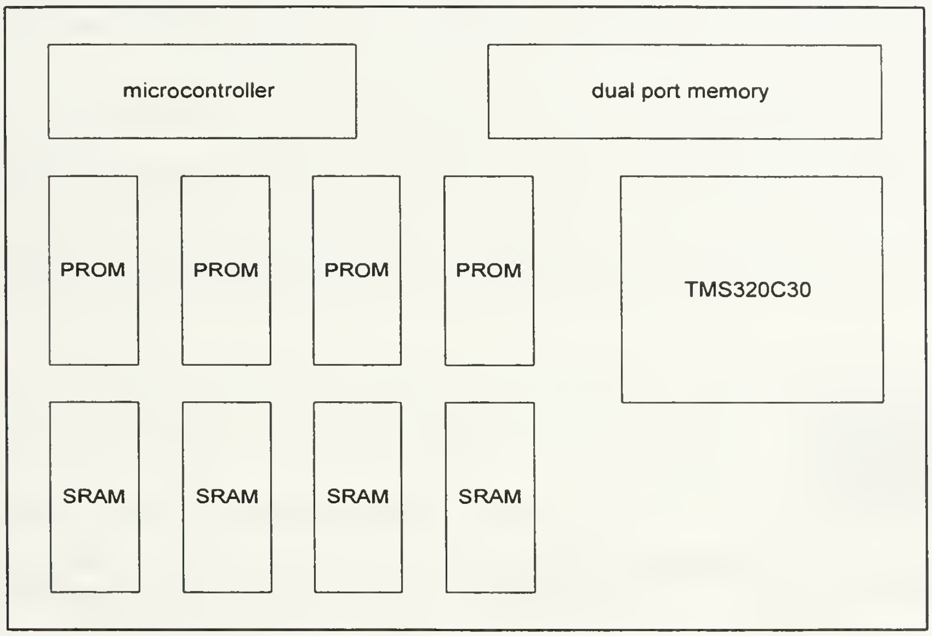

Figure 3-5, The Universal Controller CPU board showing key components

The last memory area is the dual-port memory. The IDT7130SA high-speed 1K x 8 dual-port static RAM from Integrated Device Technology was selected for use on the Universal Controller. The dual-port memory allows the DSP chip to access data that may be downloaded to the dual-port memory from the host PC. The dual-port memory is memory mapped to 100000h. Figure (3-5) shows the location of the dual-port memory on the CPU board.

c) PC Interface

The PC communicates with the universal controller board by a windows C++ program that sends data and commands through one of the communication ports to the Universal Controller. The Universal Controller contains two optical receivers. One to receive data from the host PC, and the other to receive data from another Universal Controller. Since the Universal controller must receive an optical signal, the signal from the PC must be converted to an optical signal as shown in Figure (3-6). The RS-232 port from the host computer is connected to a optical transmitter/receiver box. Pins 2, 3, 4, and 7 of the RS-232 are used and correspond to: transmitting data, receiving data, requesting to send data, and signal ground respectively.

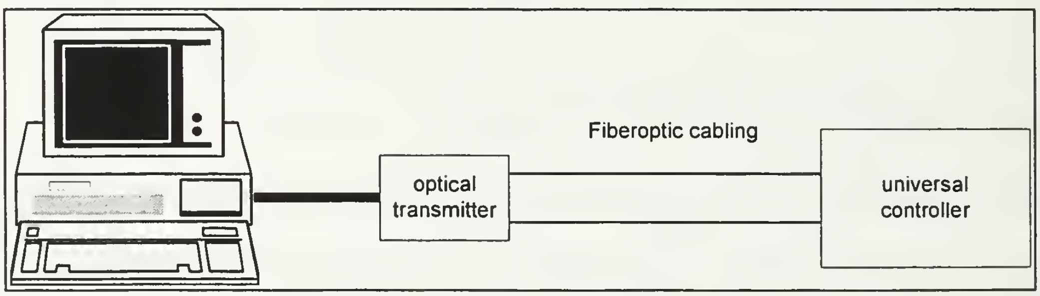

Figure 3-6, PC interface.

The optical transmitter/receiver box uses a MAX232CPE +5V RS-232 transceiver to drive the optical transmitter and receiver ports. The data is sent to the microcontroller on the CPU board, Figure (3-5). The microcontroller is Intel's 87C51FA. The microcontroller can write to the dual-port memory, interrupt the DSP processor, and send commands to the DSP processor. It is operating from a 16 MHz clock. The 8751 must

be programmed in conjunction with the C++ windows program for the interface to work. An engineer at NSWC, Annapolis has written both the C++ PC code and the 8751 assembly code to allow for limited interaction with the Universal Controller. Currently the PC can download 20 parameters to the dual port memory and send commands to start and stop the controller.

d) Analog Interface

The input/output board contains 11 Maxim 500ksps 12-bit A/D converters. The sensed voltages and currents are connected to the I/O board via twisted pair. The input range is ± 5 volts and the conversion time is 2.6 μs. The I/O board has one 20 MHz clock and a Motorola SN74LS93 4-bit binary counter that is used to convert the 20 MHz clock into a 5 MHz clock for the A/D converters. The conversion takes 13 clock cycles and one clock cycle takes 1/5MHz = 0.2 μs; therefore, total conversion time is 13(0.2 μs) = 2.6 μs. The A/D converters are set up in the ROM mode. This means that the converter acts like a fast-access memory location. There are 10 input connections and 11 A/D converters. The 11th A/D converter is used to convert a selected analog input signal to a digital signal for purposes of displaying on an LCD screen. There are 5 memory locations for the 10 A/D converters. A read to one memory location will initiate a conversion on two A/D converters and will also read the data from the previous conversion. The time between successive read operations must be greater than the sum of the conversion time (2.6 μs) and the track/hold acquisition time (.35 μs) [Ref.17]. Figure (3-7) shows the I/O board and the 11 A/D converters.

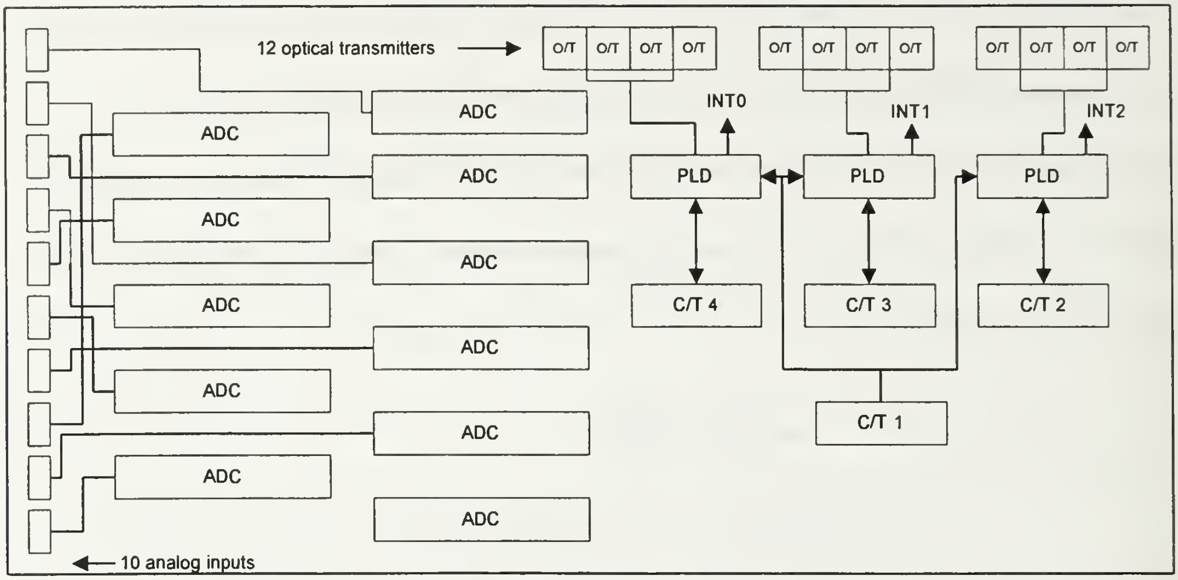

Figure 3-7, The Universal Controller I/O board showing key components

e) Timer/Counters

There are four Harris 82C54 counter timers on the I/O board as shown in Figure (3-6). The 20 MHz clock mentioned above is converted to a 10 MHz master clock signal via the SN74LS93 4-bit binary counter. This master clock signal is used by all 4 counter timers and the 3 programmable logic devices (PLD). As mentioned earlier, the Universal Controller was designed for operating the ARCP inverter which requires 12 gate signals (when controlling both the main and auxiliary switches). Figure (3-7) illustrates that there are 3 sets of 4 optical transmitters (phase a, phase b, and phase c) and that each set is controlled by 1 PLD and each PLD has a separate counter/timer and a common counter/timer connected to it. The PLD is programmed to operate the 2 main switches and the 2 auxiliary switches on a single phase of the ARCP inverter. The counter timers can be controlled by software and will be discussed in the next chapter.

Each counter timer has 3 counters, a control word register, and a data input register. For the ARCP inverter, C/T 1 is set up in rate generator mode. This mode establishes the switching frequency. The master clock is a 10 MHz clock which corresponds to T_{pulse} = 100 ns. For a 10 kHz switching frequency, the switching period is $T_{sw} = 1/f_{sw}$ = 100 μs. The 100 μs is timed by counting 1000 counts (1000 x 100 ns = 100 μs) on one of the counters in C/T 1. The other 2 counters in C/T 1 are set up to count to 1000 also, but they are delayed so that each counter in C/T 1 generates a pulse every 100 μs and each pulse is separated by 1/3 of the switching period, or 120°. C/T 1 is designated the switching frequency counter and each counter output is used by a PLD, one for each phase, separated by 120°. The PLDs will generate an external interrupt on the C30 DSP processor by each pulse generated by C/T 1, allowing data reads from the A/D converters and control calculations to be performed by the processor. The other 3 counter timers are set up to operate in hardware retriggerable one-shot mode. In this mode the PLD will trigger a counter and the output of the counter will go low for a predetermined count (whatever count is written to the counter) and then go high until triggered again. Each PLD and C/T combination along with the counts associated in each counter will drive the optical transmitters and control up to 12 gates. The detailed schematic of the PLD is contained in Appendix B.

3. Architecture Overview

As stated earlier, the Universal Controller is contained on two boards, a CPU board and an I/O board. The boards are connected by a 96-pin snap-on connector. Figure

(3-8) shows the block diagram and how the two boards are connected. The Universal Controller is powered by a Condor DC power supply model HDCC-150W-A+. The power supply is rated at 5V, 12A and ±12V, 3.4A. The 5V is connected to the CPU board and powers all the digital electronics on both boards. The ±12V is connected to the I/O board and powers the input signal buffers and A/D converters. A complete set of schematics is located in Appendix B.

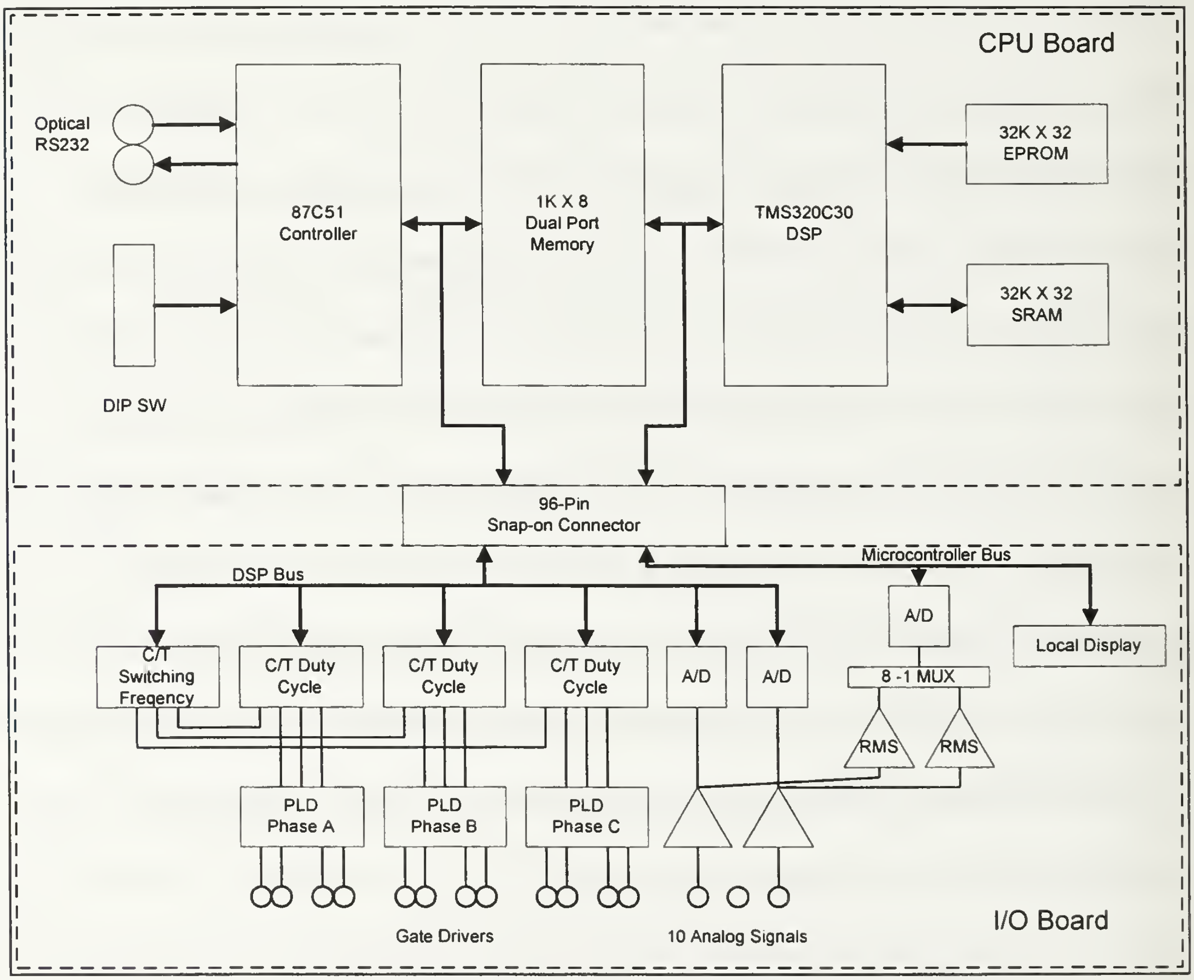

Figure 3-8, Block diagram of the Universal Controller [Ref. 15].

The hardware elements in Figure (3-8) work together to provide the control of the SSIM and the SSCM. For example, to control the SSCM, the first requirement is to burn

the control code into the EPROM. The code will need parameters such as switching frequency, reference voltage, and gain constants. These parameters are downloaded to the dual port memory from the host PC via the 8751 microcontroller. The 8751 controller will start and stop the controller by commands issued from the host PC. Once the host PC starts the controller, the DSP processor will initialize the board to run the SSCM program. Initialization consists of loading the control parameters from the slower dual port memory to the faster on-chip RAM and initializing the counter/timers according to the control parameters. Only one gate driver, one duty cycle C/T, four A/D converters and the switching frequency C/T are needed to control the SSCM; therefore, one Universal Controller can control two SSCMs. The input voltage, inductor current, output current and output voltage are sampled and converted to digital signals. The signals are used to calculate the proper duty cycle which in turn corresponds to an integer count value that must be written to the duty cycle C/T. This will produce the gate driver signal to be optically transmitted to the SSCM. The SRAM is not needed for the SSCM code but is required for sine triangle pulse-width modulation control for the SSIM A large lookup table is generated and stored in this memory location.

The TMS320C30 must be programmed to implement the specific control algorithms. The software implementation is discussed in the next chapter.

IV. DSP SOFTWARE AND FIRMWARE

A. INTRODUCTION

As discussed in Chapter III, the Universal Controller consists of many hardware components that must be programmed in order to operate. The host PC must contain software that allows it to communicate with the DSP processor, and the DSP processor must be able to communicate with all of the input/output components as well as the counter/timers. In this chapter the software and firmware requirements of the Universal Controller are introduced and discussed. The term software is defined within this document as being computer program instructions, and the term firmware is defined as the combination of a hardware device, such as the 87C51, and the computer instructions or computer data that reside as read-only software on the hardware device. The software and firmware that is required to operate the Universal Controller can be broken up into two categories: interface software and firmware and TMS320C30 firmware.

The interface programming consists of firmware and software that allow components to communicate with each other. For example, the 87C51 microcontroller must be programmed to accept parameter data from the host PC and store this data in an exact memory location within the dual port memory. This interface software and firmware is provided by NSWC.

The TMS320C30 must be programmed to do the mathematical computations required by the control algorithm. This code is assembled and is loaded into the EPROM.

It is the TMS320C30 code that contains the control algorithms developed here at NPS for controlling the SSIM and the SSCM.

B. NSWC PROVIDED SOFTWARE AND FIRMWARE

The interface hardware that is programmed by NSWC includes the 87C51 microcontroller, the programmable logic devices (PLDs), and one address decoder. The host PC must also be programmed to communicate with the Universal Controller. Since the user enters parameters and starts and stops the controller with the host PC, the primary emphasis of this section will be on the host PC software. A brief overview will then follow covering the remaining firmware.

1. Host PC Software

The host PC software is a visual C++ program written in Microsoft Windows Visual C++ version 1.5 for Windows 3.1. The program was written specifically to demonstrate the capability of the Universal Controller by controlling a specific NSWC power electronic device depending on the mode selected by the software. Since the main function of the host PC is to pass parameters to the dual port memory and start and stop the controller, the software did not need modification here at NPS. The software, as written, will suffice to pass the parameters needed for both ARCP inverter operation and buck chopper operation. Also depending on the mode you select, the host PC will signal the controller to initiate or stop the desired control algorithm. This software is installed on the research computer in the power lab. Installation instructions are included in Appendix C.

To operate the software you must open the main window. The main window will be displayed by double clicking the left button on the PEBB icon located in the PEBB working group window. The program itself is Pebb.exe and is located in the C:\Pebb\win31new directory. As illustrated in Figure (4-1), the PC can control two Universal Controllers, unit 1 and unit 2. Each unit has its own "ON" and "OFF" buttons and an "ON/OFF" display indicating the current status of the unit. Before either unit can be activated, a data link must be established. The data transfer between the PC and the controller is done through one of the PC's communication ports.

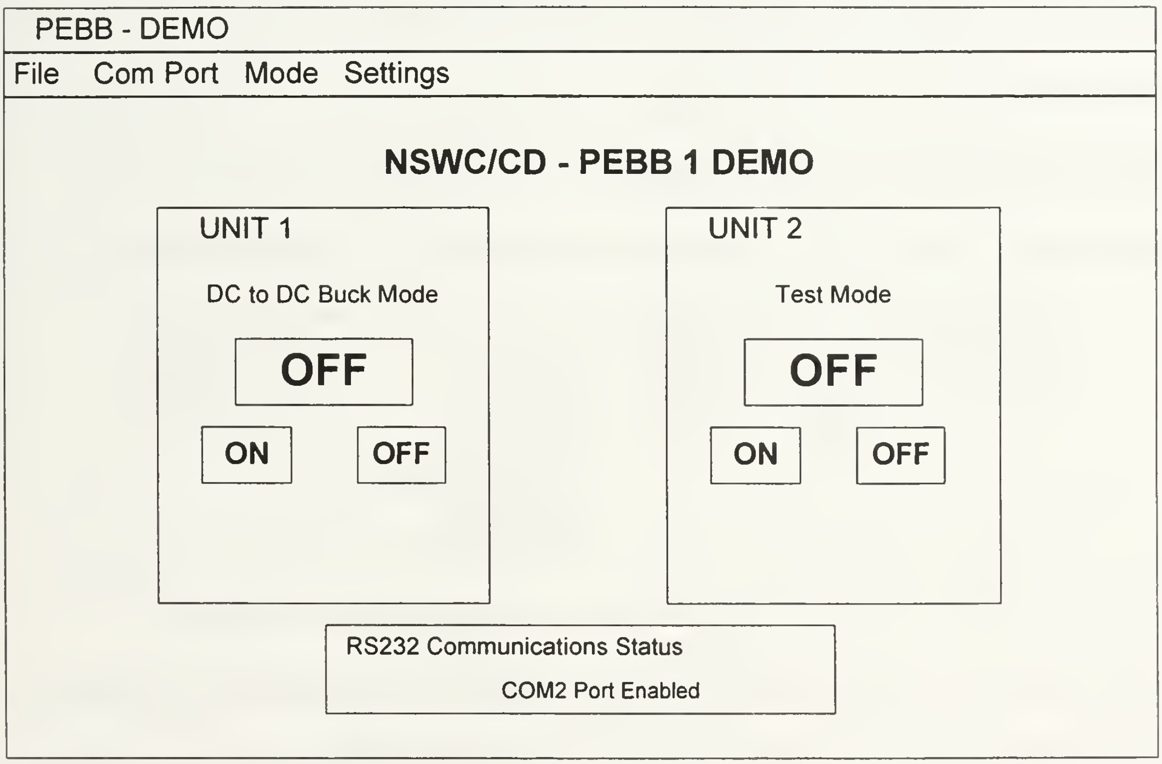

Figure 4-1, The main window for the host PC software

The Universal Controller is connected to the research computer via COM2. To tell the host PC which COM port each unit is connected to, the Com Port menu item must

be selected as shown in Figure (4-2). The demo software allows two units to be connected to one COM Port at one time. One PC can operate more than two controllers if additional COM Ports are available. Each Universal Controller has a 5-pin dip switch on the CPU board. Setting the first switch to "on" and the other four switches to "off" corresponds to unit 1. Setting up unit 1 to run the DC-to-DC buck (SSCM) is illustrated in Figure (4-2). The COM port must be selected first followed by the mode selection. The last menu selection is the Settings menu. By clicking on the UNIT 1 selection under Settings, a separate window is opened. This window is shown in Figure (4-3). From here the user can enter up to 20 parameters. It should be noted that the names associated with the parameters in the settings window are fixed and are associated with the parameters needed for the operation of the NSWC ARCP. As a consequence, in order to use the same interface software while controlling the NPS SSCMs, the variable names listed in

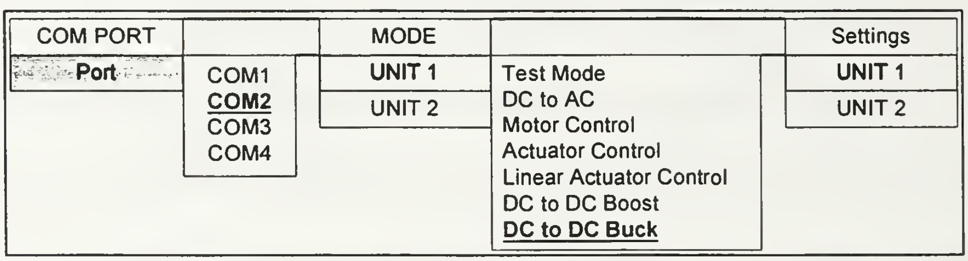

Figure 4-2, Host PC software menu settings for SSCM operation.

Figure (4-3) will necessarily need to be used to represent SSCM parameters. Therefore, this window is merely a channel to pass parameters to the Universal Controller that need to be changed on the fly. For example, to test the response to of the SSCM for different control gains, KC, KCB, and BT are used to pass these parameters to the CPU board

instead of reprogramming the PROMs. More details on the parameters entered for the SSCM will be given in Chapter V.

The parameters are passed to the dual port memory by clicking on the “OK” button in the settings window. Only after the parameters have been passed to the CPU may the controller be turned on. The red LED light on the CPU board will illuminate to indicate that the CPU received the “turn on” command by the host PC.

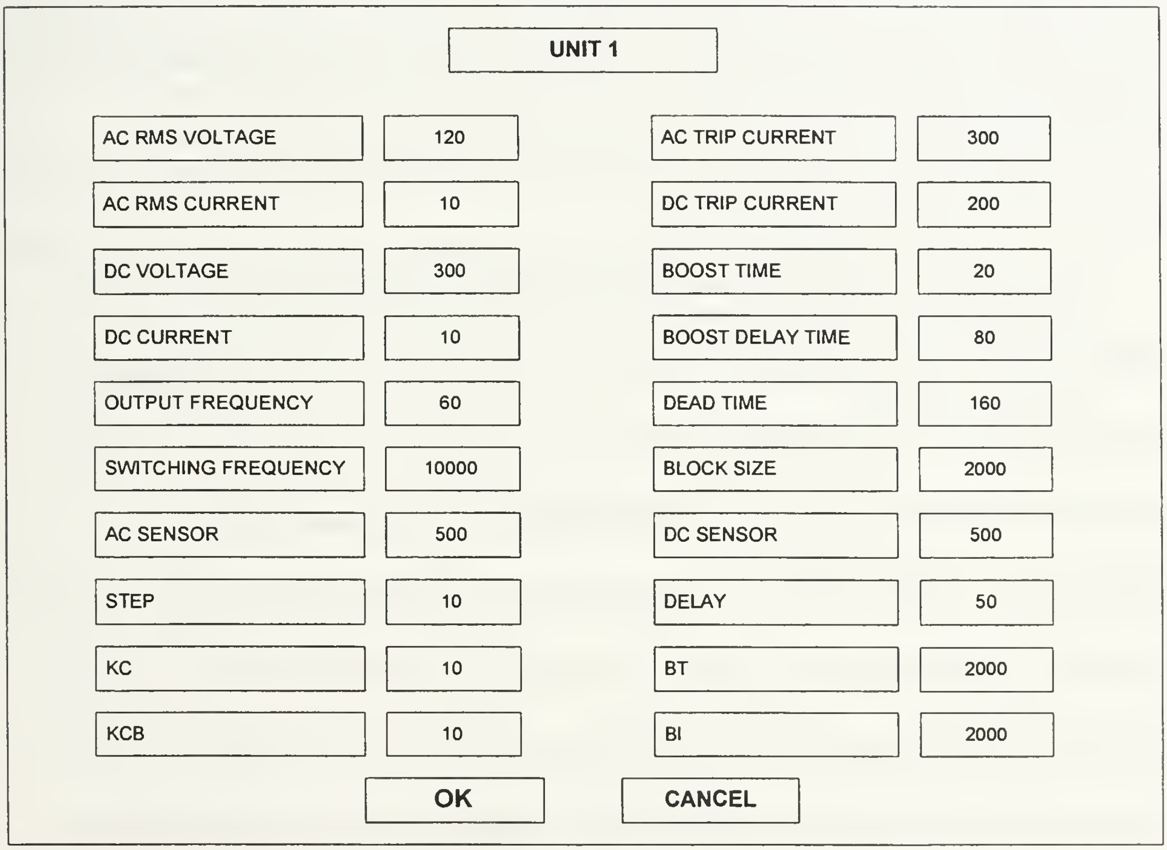

Figure 4-3, Unit 1 settings window.

Other modes listed in Figure (4-2) are modes that correspond to the NSWC TMS320C30 firmware. With the NSWC PROMs installed, selecting the test mode will operate the ARCP inverter open loop. The DC to AC mode is specifically designed to

operate the NSWC ARCP inverter closed loop. The motor control, actuator control, linear actuator control, and DC to DC boost modes are modes to operate specific NSWC devices for PEBB demonstrations.

2. Microcontroller Firmware

The microcontroller must be programmed in order to operate the Universal Controller. The 87C51 has 4K bytes of on-chip PROM to store the program. This means that rather than using an external PROM chip, the program can be burned directly into the 87C51. The instruction set, assembly language and the use of the 8051 assembler (ASM51) are described in reference [18].

The purpose of the microcontroller is to interface between the host PC and the TMS320C30 DSP microprocessor as described in Chapter III. This is accomplished in the following way. First the 87C51 checks to see what address is selected on the 5-pin dip switch. For example, if the switch was configured to correspond to unit 1 (as explained above), data sent to the controller from the host PC for unit 2 would be ignored by the CPU board with a unit 1 address. If data being sent from the host PC is for unit 1, the microcontroller will accept the data and process it. The microcontroller will store the parameters from the unit settings window in the dual port memory to be read by the DSP microprocessor later. The 87C51 will also interrupt the DSP processor. The interrupts pass commands to the DSP microprocessor to either start or stop the execution of the desired control algorithm.

3. Programmable Logic Device

There are three PLDs on the I/O board. These devices contain the logic to operate the gate drivers connected to the optical transmitters. Each PLD controls four optical transmitters, one for each gate on a single phase of the ARCP. The NPS ARCP does not require all four optical transmitters since it has its own auxiliary circuit controller; therefore, only two of the four optical transmitters per phase are used to control the main switches. The PLD logic produces gate signals as illustrated in Figure (4-4) with a 1.2 μs delay time between them. This delay time is referred to as "dead time" and is required by the outer controller block of the ARCP as discussed in Chapter II. This dead time prevents both upper and lower switches from being "on" at the same time.

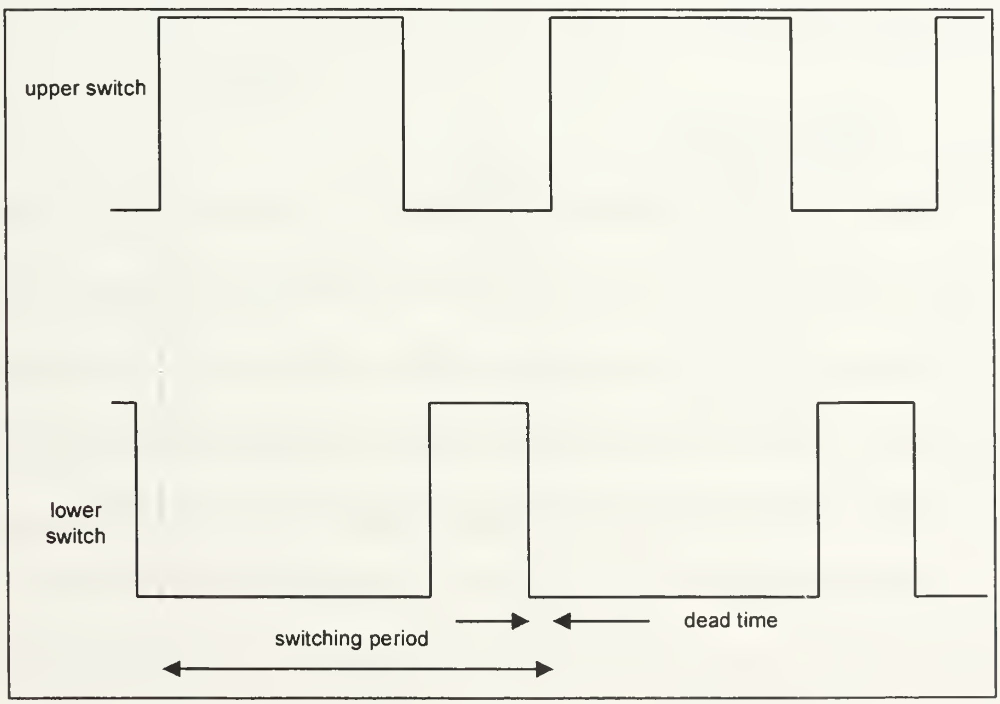

Figure 4-4, The signal produced by the PLD logic.

All three PLDs have the same logic burned in them. As shown in Figure (3-7), each PLD has its own counter/timer connected to it and one shared counter/timer connected to it. The shared counter timer is called the switching frequency timer (this will be explained later). Each PLD generates a switching period using the shared counter/timer and a duty cycle using the dedicated counter/timer. For example, if the desired switching frequency is 20 kHz the switching frequency timer must have an integer value of 500 in its counter register. This is because each count takes 100 ns and 500 x 100 ns = 50 μs which is the desired switching period for 20 kHz. The duty cycle is determined by the count that is stored in the second counter register (each counter/timer has 3 counters) on the dedicated counter/timer. If this count is 250, a 50% duty cycle will be produced. More details will be documented in Chapter V.

4. Address Decoders

In addition to the PLDs required for controlling the optical transmitters, a fourth PLD is needed for address decoding. The EPM5016 is used for reading memory from the four EPROM chips and four SRAM chips. The TMS320C30 has a 32-bit data bus and the EPROM and SRAM contain 8-bit memory locations. The EPM5016 is programmed so that one read by the microprocessor will produce a 32-bit word from either the four SRAMs or the four PROMs.

C. TMS320C30 FIRMWARE

The TMS320 family of DSPs are supported by a complete set of software development tools, including an optimizing C compiler, an assembler, a linker, an

archiver, and a software simulator. Figure (4-5) illustrates the software development flow for the TMS320C30. The highlighted area shows the development path negotiated for this thesis.

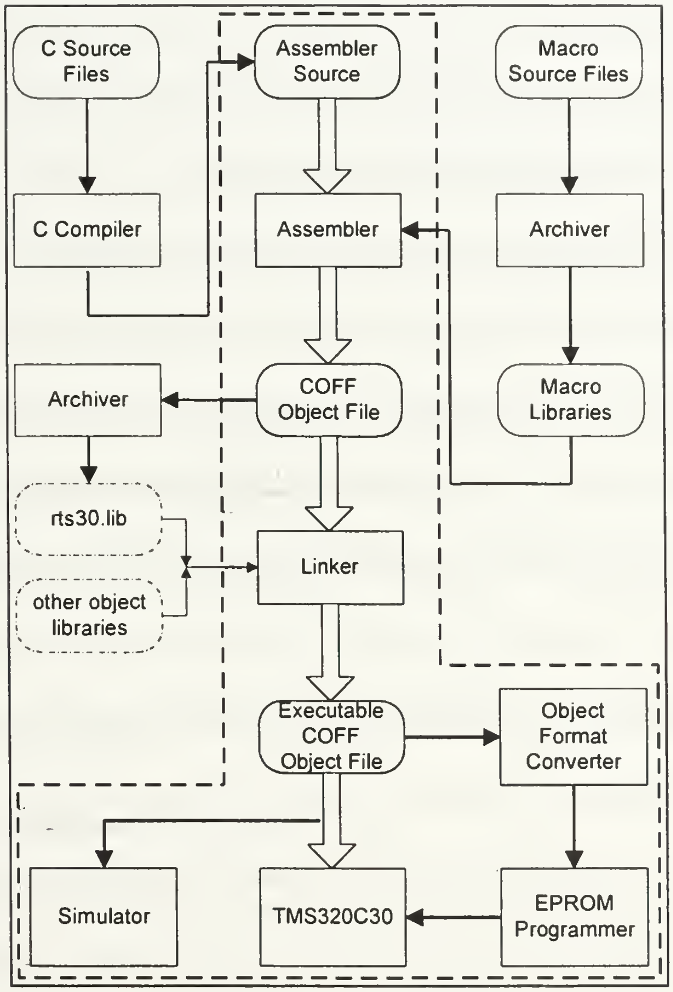

Figure 4-5, TMS320C30 software development flow [Ref. 16].

The tools shown in Figure (4-5) perform key functions in the software development flow. A brief description of each tool is given below. Detailed information is found in References [19, 20, and 21].

- The **C compiler** accepts ANSI standard C source code and produces TMS320C30 assembly language source code. The compiler includes an interlist utility which interlists the C source statements with the assembly language output.
- The **assembler** translates the assembly language source files into machine language Common Object File Format (COFF).
- The **archiver** combines a collection of files into a single file called a library. The archiver can also be used to modify the existing library by adding, replacing, extracting, or deleting members. The object library rts30.lib is included with the C compiler. It contains standard runtime-support functions, compiler utility functions, and math functions that can be called from C programs.
- The **linker** accepts COFF object files and object libraries as input and combines them into a single executable object module.
- The **object format converter** converts the COFF object file into either TI-tagged, Intel hex, Intel word, or Tektronix object format. The converted file is then downloaded to an EPROM programmer.
- The **simulator** is a software program designed to simulate the executable COFF object file produced by the linker. The simulator allows for on-screen editing and shows continuous updates as you step through the code. The simulator also features multiple windows to view the assembly code, CPU

registers and multiple memory locations. A complete user's guide is located in the NPS power lab [Ref. 21].

As shown in the highlighted portion of Figure (4-5), the software development for this research started at the assembly language level. It might seem that given the choice of writing source code in C versus assembly language, one would clearly choose writing the code in C. The reason for starting at the assembly language level is twofold. First of all, NSWC did all their coding in assembly language and although this code did not have the control algorithms for the SSCM or the SSIM, it did contain all of the initialization code (explained below) for the Universal controller. The time constraint place on this project did not allow for the reinvention of the wheel, so much of the NSWC code was used for the initialization of the board. Secondly, since there is no user's manual or documentation for the Universal Controller, looking at the assembly code and the schematics was the only way to understand how the board operates on the physical level.

1. Texas Instruments Assembly Code

The TMS320C30 instruction set contains 113 instructions. Most require one clock cycle to execute. The instruction set can be organized into the following five functional groups:

- Load-and-store
- Two-operand arithmetic/logical
- Three-operand arithmetic/logical
- Interlocked operations

• Parallel operations

This is a very powerful combination of instructions. As it happened, only instructions contained in the first three groups were used for this research. Reference [16] contains a detailed description of all 113 instructions.

The program code is written so that more than one control algorithm can be implemented without reprogramming the PROMs. For example, the NSWC host PC software contains a menu of different modes to choose from and each mode required a different control algorithm. Each control algorithm was coded on the same PROM as a subroutine. Figure (4-6) shows the general flow diagram of the source program. The board is reset and initialized by turning on the board's power supply. Once the board is initialized, it waits for an interrupt from the 87C51 to tell it what to do.

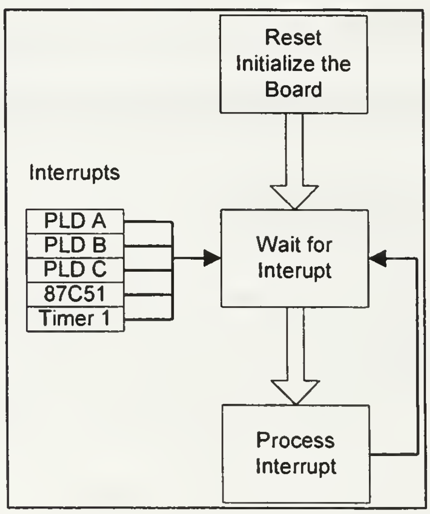

Figure 4-6, General flow diagram of the source program

The initialization of the board can be broken down into five categories. A brief overview of each category is described as follows:

a) Initialize Bus Control

The TMS320 has two external interfaces, the primary bus and the expansion bus. Both primary and expansion buses consist of a 32-bit data bus and a set of control signals. The primary bus has a 24-bit address bus and the expansion bus has a 13-bit address bus. The primary bus is used to read instructions from the PROM and read and write to the SRAM and dual port memory. The primary bus must generate one wait state to read from this slower memory. The primary bus also must also be set to allow for 1 M bank switching. The bank switching feature of the primary bus provides a period of time for allowing the PROM and SRAM to release the bus during the dual port memory reads. The primary bus is programmed to operate as discussed above by writing 428h to the control register for the primary bus which is located at memory location 808064h.

The expansion bus is connected to all of the A/D converters and counter/timers. To allow for the delay associated with reading and writing to these devices, 2 wait states are required to be generated by the expansion bus. The expansion bus is programmed to operate as discussed above by writing 48h to the control register for the expansion bus which is located at memory location 808060h.

b) Initialize stack pointer

The stack pointer is a 32-bit register that contains the address of the top of the system stack. This address is the location in memory that register values

are pushed to during an interrupt. The stack pointer is initialized to 809F00h which is located in one of the internal RAM blocks.

c) Initialize counter/times

As introduced in Chapter III, there are four counter/timers situated on the I/O board. Figure (3-7) illustrates that each PLD has its own counter/timer (C/T) connected to it along with C/T1. Also recall that each C/T has three counters and these counters can be configured to operate in different modes. C/T1 is called the switching frequency timer. The switching frequency counter is programmed so that all three counters operate as a divide by N counter (mode 2 as referred to in Ref. [22]). Figure (4-7a) shows the operation of mode 2. In this mode, when a count N is written to the counter control register, the output of the counter stays high for N-1 counts (recall that 1 count is 100 ns) and goes low for the last count before repeating the cycle. The PLDs are programmed to generate an interrupt when the output of the counter goes low and these interrupts will be used to implement the control algorithm.

The other three C/Ts are programmed as hardware retriggerable one shot timers (mode 1 in Ref. [22]). While all three counters on each of these C/Ts are set up in mode 1, only counter 2 is used to generate the duty cycle for both the SSCM and SSIM. Figure (4-7b) shows the operation of mode 1. The output of the mode 1 counter is “high” until the gate input of the counter senses a rising edge. This triggers the counter and causes the output of the counter to go “low” for a count of M. M can be written to the counter at

any time and can be changed at any time; however, writing a new M value during the low output interval will not change the current count.

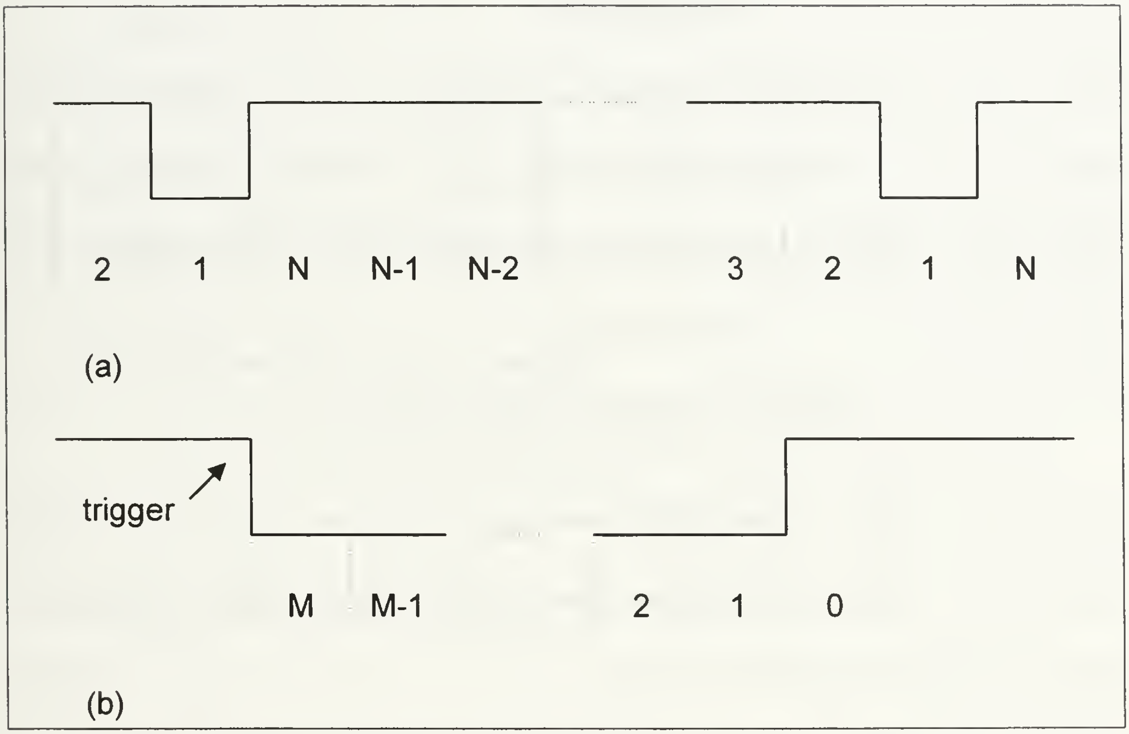

Figure 4-7, (a) Mode 2 divide by N counter. (b) Mode1 hardware retriggerable one shot.

The counter/timers are configured by calling a subroutine called init_ct. An example of this will be illustrated in Chapter V. This subroutine is listed in Appendix D.

d) Initialize memory pointers

There are eight 32-bit auxiliary registers (AR0 - AR7) that can be accessed by the CPU. The primary function of the auxiliary registers is the generation of 24-bit addresses. Part of the board initialization is to set up four of these registers as

permanent memory pointers to frequently accessed memory blocks. Table (4-1) shows the memory mapping for these registers.

Register	Memory Block	Address
AR3	Pointer for Internal Memory Block 1	809C00 hex
AR4	Pointer for Dual Port Memory	100000 hex
AR5	Pointer for Internal Memory Block 0	809800 hex
AR7	Pointer for SRAM	80000 hex

Table 4-1, Initialized address pointers.

e) Enable 87C51 interrupt

After all of the above initialization has taken place, the following two lines of code enable the 87C51 interrupt.

```
LDI     0008H,IE
OR      02000H,ST
```

The first statement sets the 87C51 interrupt bit in the TMS320C30's Interrupt Enable Register. Table (4-2) shows the 9 least significant bits of the IE register and the associated interrupt for each. By writing 0008H to this register, bit 3 becomes set. The second statement sets the Global Interrupt Enable bit (bit 13) in the TMS320C30's Status Register. This bit must be set or the C30 will not respond to any interrupts, even if they are enabled in the IE register.

Once the 87C51 interrupt has been enabled, the initialization process is complete. Appendix D contains the complete listing of the assembly code for operating the SSCM and SSIM.

Bit	9	8	7	6	5	4	3	2	1	0
Interrupt	timer 1	timer 0					87C51	PLD C	PLD B	PLD A

Table 4-2, Least nine significant bits of the IE register and their associated interrupt.

2. Texas Instruments C Compiler

As stated earlier, the software development tools include a C compiler that produces TMS320C30 assembly source output. This allows the programmer to write the entire program in C or intermix C functions and subroutines with an assembly language program.

The C compiler, along with all the TMS320 development software, is installed on the DOS computer #M051273 in BU-114. The compiler is called up by typing cl30 on the command prompt in the DSPTOOLS directory. Complete instructions on how to compile TMS320 C programs is contained in Reference [19].

Now that the Universal Controller has the software and firmware required to initialize the board as explained above and the ability to receive commands and parameters from the host PC, a program must be developed to implement an SSIM or SSCM control algorithm. The development of this code is discussed in the next chapter.

V. DSP CONTROL IMPLEMENTATION

A. INTRODUCTION

After the Universal Controller is initialized as described in Chapter IV, the TMS320C30 waits for an interrupt from the host PC to communicate the desired operating mode. Once it knows what mode to operate in it will perform the desired control algorithm. The mode setting is nothing more than a pointer to a subroutine that implements the control algorithm. This chapter first includes documentation on how the DC-to DC converter control algorithm was implemented using the Universal Controller. Secondly, a description of NSWC's ARCP open-loop control implementation and an outline of the modifications required to implement closed-loop control are presented.

B. DC-TO-DC CONVERTER DSP CONTROL IMPLEMENTATION

The following control algorithm, introduced in Chapter II, is the control law implemented on the 3 kW converters located in the Power Systems Laboratory at NPS:

$$\mathrm{d(t)} = \mathrm{Dss} - \left(\mathrm{h_v} + \mathrm{h_n}\int dt\right)\left(v_0 - v_{ref}\right) - \mathrm{h_i}\left(i_L - i_0\right) \tag{5-1}$$

where,

d(t) = time-varying duty cycle	Dss = steady-state duty cycle = $V_{ref}/V_{in}(t)$
h_v = voltage gain	h_n = integrator gain
v_0 = buck output voltage	h_i = current gain
v_{ref} = reference voltage	i_0 = load current
i_L = inductor current	$V_{in}(t)$ = input voltage

The only difference between Equation (5-1) and Equation (2-4) is the $i_o/10$ term. This term is required for paralleling two units and since the 3 kW converters are stand-alone converters they will not be required to operate in parallel.

Equation (5-1) must be implemented digitally. To do this, the following discrete terms must be defined:

- **V_{in}[n], V_{out}[n], i_{out}[n], and i_L[n]** are the input voltage, output voltage, output current, and inductor current respectively sampled during switching period [n].
- **V_{ref}** is the desired output voltage and can be entered into the program via the host PC Settings window.
- **Dss[n]** is the steady-state duty cycle calculated during the current switching period using the following equation:

$$Dss[n] = \frac{V_{ref}}{V_{in}[n]} \tag{5-2}$$

- **v_d[n] and v_{dint}[n]** are the voltage error and the integral of the error calculated during switching period [n]. The voltage error is simply the output voltage minus the reference voltage as shown in Equation (5-3). The integral of the error will be examined below.

$$V_d[n] = V_{out}[n] - V_{ref} \tag{5-3}$$

- **v_d[n-1] and v_{dint}[n-1]** are the voltage error and the integral of the error calculated during the previous switching period, [n-1].
- **d[n+1]** is the duty cycle for the next switching period.

The integration term is calculated using the trapezoid integration method. While other integration methods can be used, this method was selected because it appeared to be the simplest numerically. Figure (5-1) shows two consecutive discrete values of v_d. The

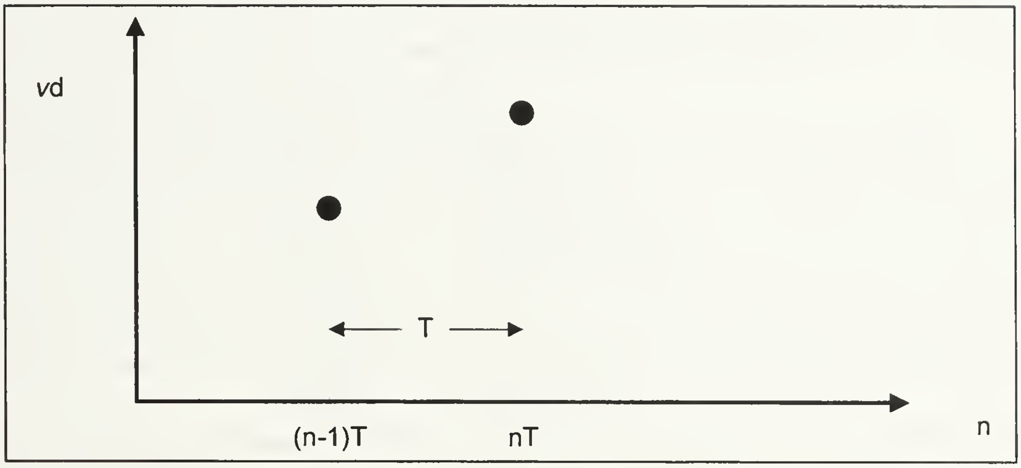

Figure 5-1, Trapezoid integration.

approximation of the integral is established by connecting the two points with a straight line and identifying the area of the resulting trapezoid and then adding this quantity to the prior integration error. This is illustrated in the following equation:

$$v_{dint}[n] = v_{dint}[n-1] + \frac{T}{2}\left(v_d[n-1] + v_d[n]\right) \tag{5-4}$$

Using the above approximation for the integral term, Equation (5-1) becomes:

$$\begin{aligned} d[n+1] &= Dss[n] - h_v\left(v_{out}[n] - V_{ref}\right) \\ &- h_n\left\{\frac{T}{2}\left(v_d[n-1] + v_d[n]\right) + v_{dint}[n-1]\right\} - h_i\left(i_L[n] - i_{out}[n]\right) \end{aligned} \tag{5-5}$$

It is clear from the above equation that only the integration term requires two memory states in its calculation. All of the other terms require only the current sampled values.

1. Control of One DC-to-DC Converter

Controlling one buck converter requires the following I/O board hardware devices to be active: two PLDs, two counter/timers, four A/D converters, and one optical transmitter as illustrated in Figure (5-2). The A/D converters are required to convert the sensed voltages and currents to digital form. The PLDs and counter/timers generate the switching frequency and duty cycle to be transmitted by the optical transmitter to the gate driver circuit on the buck converter. This hardware is configured after the C30 receives the "DC-to-DC Buck" mode command from the host PC.

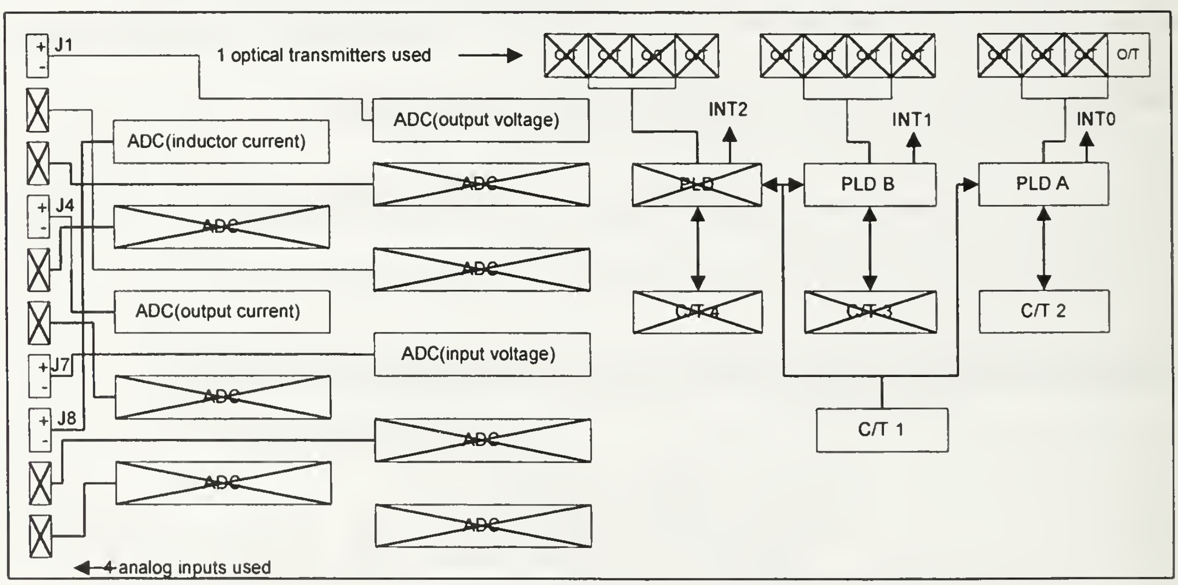

Figure 5-2, Active hardware components on I/O board for controlling one buck.

Chapter IV documented how the Universal Controller was initialized. The last step in this initialization process was to enable the 87C51 interrupt. This interrupt is used to communicate to the controller what mode to operate in. By following the procedure developed in Chapter IV, selecting the DC-to-DC Buck mode will interrupt the C30 and execute the following interrupt service routine:

```
LDI     @dp_cint,IR0
LDI     *+AR4(IR0),R0
CALL    read_cmd
ANDN    mask_int3,IF
```

These four lines of code are what start the controller.

The first two lines clear the 87C51 interrupt caused by the host PC. IR0 is an index register used for indirect addressing. The contents of memory location dp_cint (which stands for dual port clear interrupt) is loaded into IR0. Recall that AR4 is the pointer for the dual port memory (Table 4-1). By reading the memory location *+AR4(IR0) the interrupt to the 87C51 caused by the host PC will clear, which will allow for future host PC interrupts. The CALL instruction invokes the subroutine read_cmd. This is the subroutine that reads the dual port memory location that contains the mode selected (for instance DC-to-DC Buck) and appropriately configures the I/O board (as shown in Figure (5-2)). After the read_cmd subroutine is finished, the last line of code clears the interrupt flag to allow for future 87C51 interrupts.

The read_cmd subroutine retrieves an integer from the dual port memory that corresponds to the desired mode entered in from the PC. For example, when the DC-to-

DC Buck mode is selected, an integer value of 10 is written to memory location 100001h within the dual port memory. The program reads this value from the dual port memory and uses this number to branch to the location in the program that configures the controller for DC-to-DC Buck mode.

The location in the program for DC-to-DC Buck mode is labeled "cmd10:" and a listing can be found along with the entire program in Appendix B. Figure (5-3) illustrates the program flow starting with the 87C51 interrupt. It is this portion of the code that configures the I/O board according to Figure (5-2).

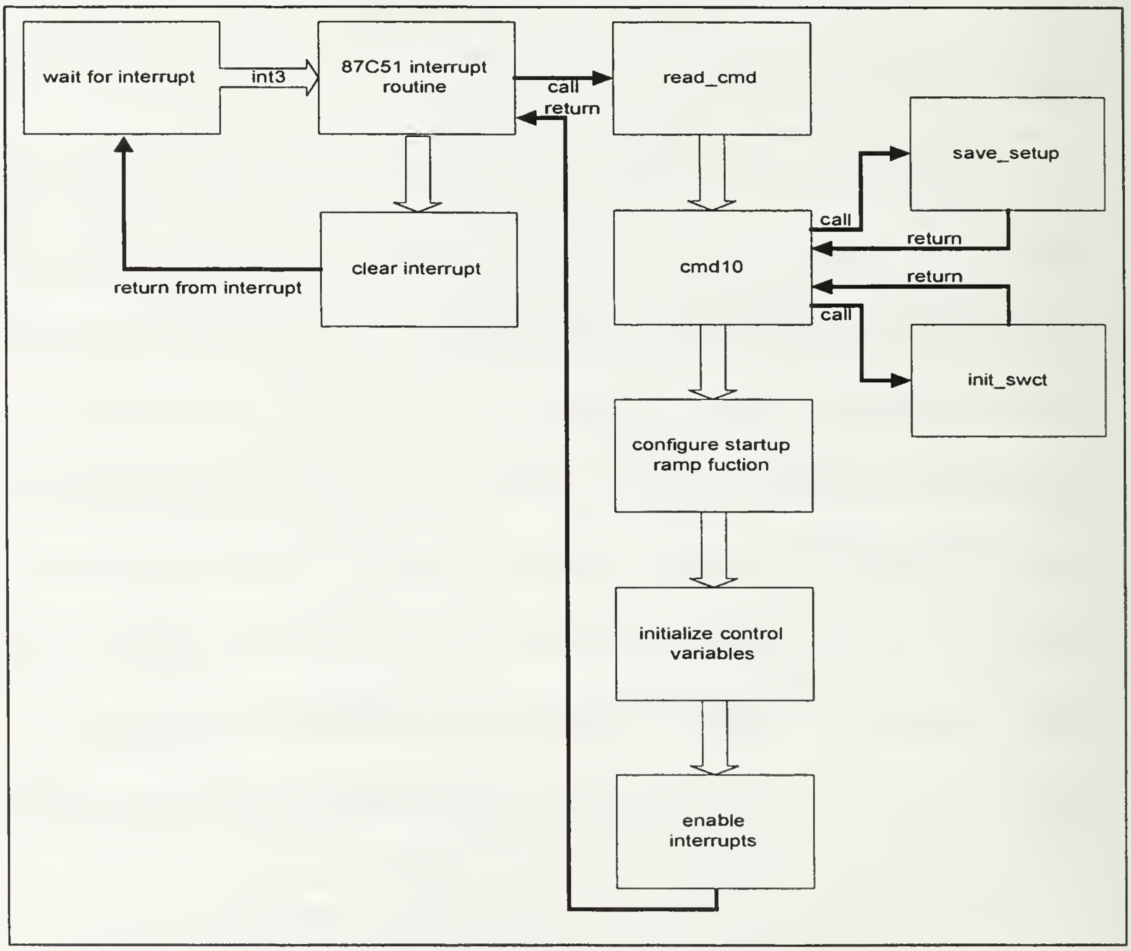

Figure 5-3, Program flow diagram for 87C51 interrupt routine.

The first task includes loading the parameters that were stored in the 8-bit word dual port memory locations (these are the parameters entered from the Settings window on the PC software) into the C30 internal 32-bit memory block. This is done in the subroutine save_setup. This subroutine also calculates the switching period in terms of number of counts it takes the switching frequency timer to count up to the switching period time. This is accomplished by dividing the desired switching period by the period of one count (100 ns) on the counter/timer. For example, if the desired frequency is 20 kHz, then the switching period in counts is (50 μs)/(100 ns) = 500 counts. This value is stored in memory location tms_swp. For reasons explained below, the switching period count is also divided by two and stored in memory location tms_swp_120.

The next task performed by the program (shown in Figure (5-3)) is to call subroutine init_swct. This subroutine starts the switching frequency counter timer, C/T1, by storing the switching period count (tms_swp) in the counter(0) register and counter(1) register. Figure (5-4) depicts the configuration of this process. Before loading counter(2) with tms_swp, the processor waits for a count of tms_swp_120 which is half of tms_swp as explained above. The reason for this delay will become clear later. Counter(0) output is inputted into PLD A and causes PLD A to interrupt the C30 once every cycle (int0). Likewise, counter(1) is inputted into PLD B which causes interrupt 1 (int1) to occur between the PLD A interrupts. These interrupts will not occur until the int0 and int1 have been enabled at the end of the read_cmd subroutine.

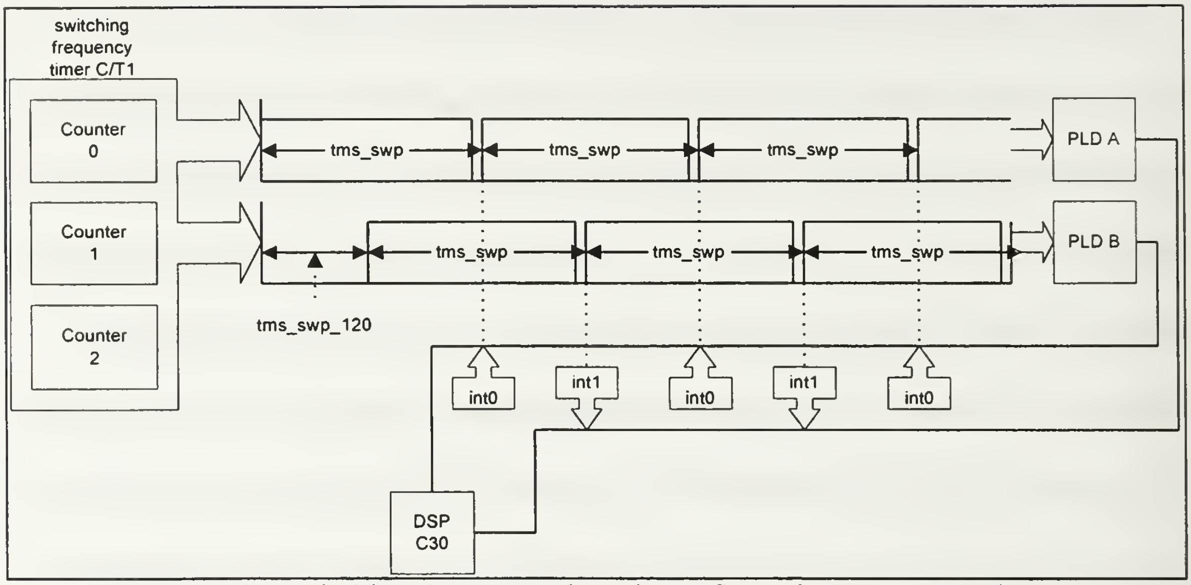

Figure 5-4, The switching frequency timer is configured to cause PLD interrupts.

Following the program flow diagram in Figure (5-3), the next section of code configures the startup ramp function. This function implements a linear increase in the reference voltage to avoid large current and voltage oscillations that would result from a large step in V_{ref}. There are two parameters that are entered from the Settings window that correspond to this function. These parameters are STEP and DELAY as shown in the Settings window in Figure (5-5). The Step parameter is the number of voltage steps desired for the reference voltage ramp from 0 to V_{ref}. For example if the reference voltage (desired output) is 208 volts and the STEP value is 50, the reference voltage would start at 0 volts and step up by (208 modulus 50) = 4 volts every DELAY time interval. As implied in the previous statement, the DELAY parameter is the amount of time the calculated command voltage stays at a given level. This is accomplished by using one of the C30's internal timers (timer0) which causes an interrupt to occur after each delay count. The delay count is calculated by multiplying the DELAY parameter by 100 and

then storing this in the period register of timer0. Each count for the internal timer takes 100 ns (T_c) so if the desired ramp-up time is 5 seconds and the number of desired steps is 50 the DELAY value entered at the Settings window should be:

$$\mathrm{DELAY} = \frac{\text{ramp up time}}{100(T_c)(\mathrm{STEP})} = \frac{5}{100 * 100 * 10^{-9} * 50} = 10000 \qquad (5\text{-}6)$$

Once V_{ref} reaches the desired reference voltage the timer0 interrupt is disabled.

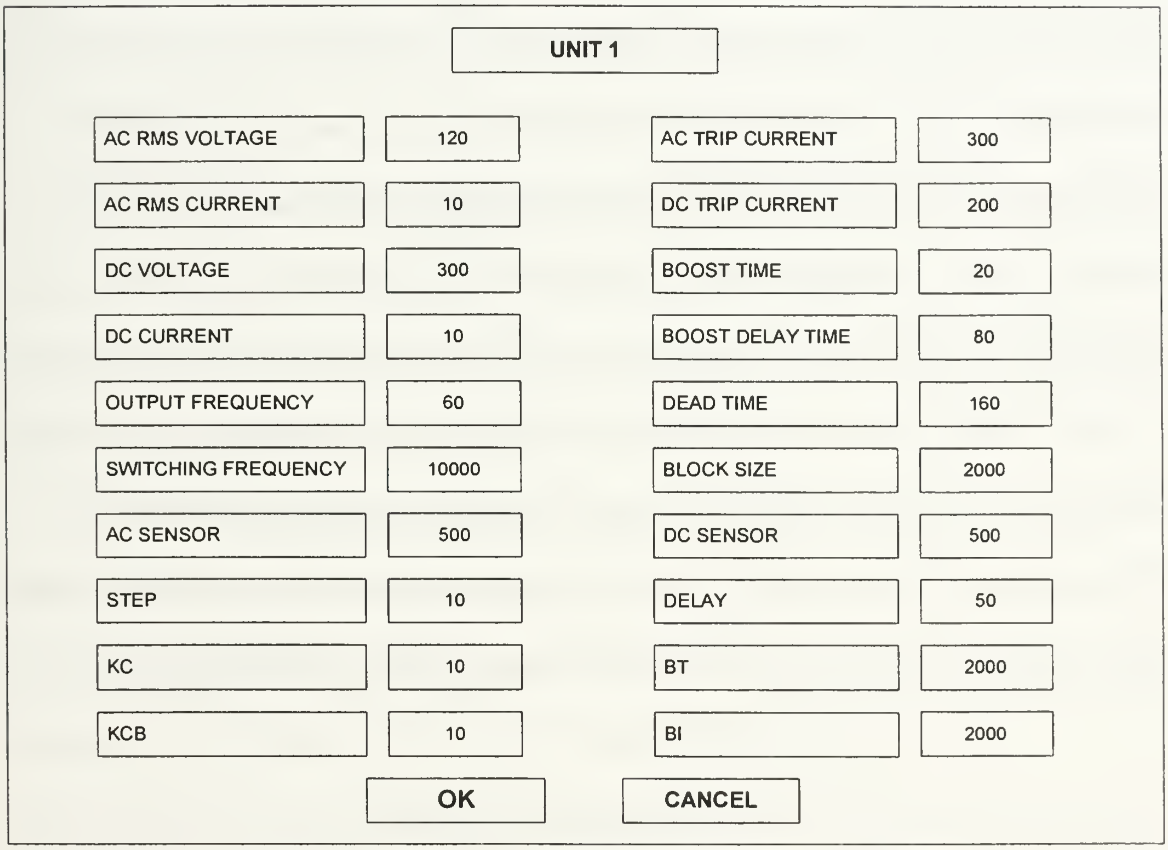

Figure 5-5, The Settings window on PC software.

The next step in the listed program flow is to initialize the control values. The following parameters from the Settings window must me modified since the software limits these values to be integers: DC VOLTAGE, AC SENSOR, DC SENSOR, KC,

KCB, and BT. DC VOLTAGE is used to enter the reference voltage which is stored as an integer and must be changed to a floating point value. DC SENSOR and AC SENSOR are the conversion factors needed to convert sampled voltages and currents to actual values. This value depends on the scaling factor of the sensor. For example, the voltage sensor for the 3 kW converter has a voltage divider of 100:1 and the current sensor implements a scaling factor of 10:1. These ratios insure that the limits of the A/D converter inputs are not exceeded given that the maximum input voltage does not exceed 500 volts and the maximum current does not exceed 50 amps. Recall from Chapter II that the output of the A/D converters produce 12-bit two's complement words. Therefore, a 4.9975 volt input corresponds to a 011111111111_b output, and a -5 volt input corresponds to a 100000000000_b output. By dividing the output of the A/D converter by $2^{11} = 2048$, it will yield a value between [-1,1] which will be the normalized value of the actual voltage or current. Multiplying this by 5 will give the voltage at the input of the A/D converter and then multiplying this product by the sensor's scale factor will yield the actual voltage. The values that must be entered for the 3 kW converter are therefore:

$$\text{DC SENSOR} = 5 * 100 = 500 \tag{5-7}$$

$$\text{AC SENSOR} = 5 * 10 = 50 \tag{5-8}$$

The remaining 3 parameters, KC, KCB, and BT, are the gains used in Equation (5-5). These parameters must be modified since the PC software will allow only integer values between 1 and 60000. A derivation of the small-signal pole locations and the resultant control gains is presented in Reference [7]. The desired gains and the required integer values that must be input at the Settings window are listed in Table (5-1). The

last column of Table (5-1) contains the scale factor that is used to convert the integer value to the desired value. The integration term (T/2) in Equation (5-5) is collapsed into the gain h_n and stored in memory location hn. KCB and BT are stored in memory locations hv and hi respectively.

Gain	Parameter	Desired Value	What is Entered	Factor
h_n	KC	1.7333	17333	$(T/2)10^{-4}$
h_v	KCB	0.0008686	8686	10^{-7}
h_I	BT	0.0105	105	10^{-4}

Table 5-1, Gains derived in Reference [7] and its associated Settings parameter.

The last item on the program flow diagram illustrated in Figure (5-3) is the enable interrupts block. As illustrated in Table (4-2), writing 010aH to the Interrupt Enable (IE) register will enable the interrupt PLD B, 87C51, and timer0.

The 87C51 interrupt in now complete and the Universal Controller is ready to start the ramp-up and the closed-loop control (Equation (5-5)). Before examining how this control is implemented and the duty cycle varied, some discussion is required on how the duty cycle is produced.

Figure (5-2) illustrates that one optical transmitter is used transmit the duty cycle to the converter and this duty cycle is generated by the combination of PLD A, C/T2, and C/T1. The PLD must generate an inverted duty cycle to drive the optical transmitter so that it will transmit light for the portion of the duty cycle that requires the switch to be closed. This prevents the converter from operating (closing the switch) if the Universal Controller loses power. With this in mind, Figure (5-6) illustrates how the count in the

counter(2) register of C/T2 generates the "OFF" portion of the duty cycle. For example, if a 75% duty cycle is required and the switching frequency is 20 kHz, then a count of (1 - 0.75)x(500 counts) = 125 counts must be written to the counter(2) register.

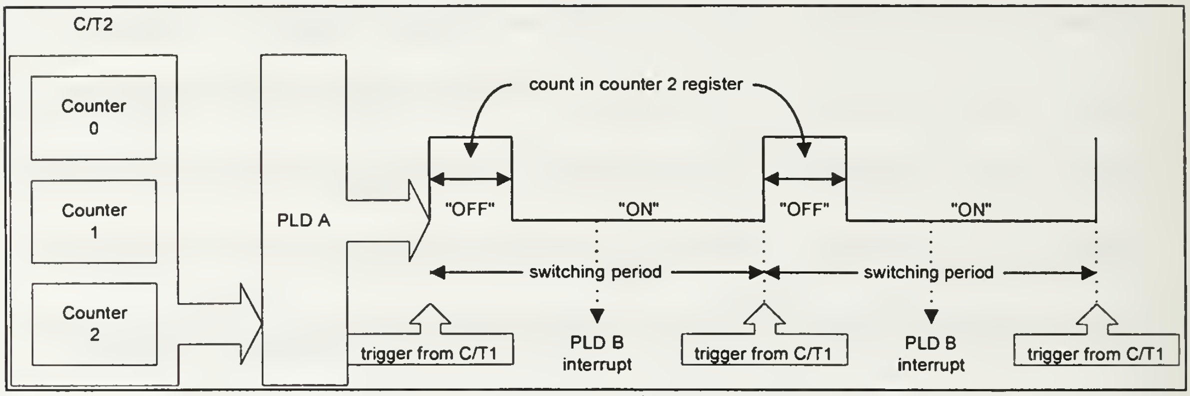

Figure 5-6, PLD A generating the control signal

The duty cycle is varied by changing the count value in the counter(2) register. The control algorithm described in Equation (5-5) is implemented by making use of the PLD B interrupt. Figure (5-6) shows the point in the duty cycle where the PLD B interrupt occurs. The reason for placing the PLD B interrupt in the middle of the duty cycle is based on two considerations. The first consideration is the time needed to do the required calculations. By placing the interrupt further to the left (earlier in the cycle), it will allow for more duty cycle calculation time (the duty cycle must be calculated before the start of the next period, see Figure (5-6)). For example, given a switching frequency of 20 kHz and PLD B interrupt as shown in Figure (5-6), the time given to calculate the new duty cycle (d[n+1]) is limited to < 25 μs. The second consideration is when to sample the voltages and currents. It is best to sample these signals as far away from the switching instant as possible to avoid corrupting the samples with noise. Although the

duty cycle can vary between [0,1], the steady-state duty cycle is ≈ 70% so that most of the switching time will vary between [.5,1]. By placing the interrupt further to the right (later in the cycle), it will reduce the chance of sampling during a switching event. After taking these two considerations into account it was decided to place the PLD B interrupt in the middle of the switching period as shown in Figure (5-6). This interrupt time can be adjusted by changing the parameter tms_swp_120 as discussed earlier.

The PLD B interrupt will occur every 50 μs (for a 20 kHz switching frequency) and it will trigger an interrupt service routine that implements Equation (5-5). Figure (5-7) illustrates the PLD B service routine. As mentioned earlier, the routine must not take more than 25 μs to implement.

The first task indicated in the flow diagram in Figure (5-7) is to sample the voltages and currents. The voltage and current waveforms that are to be sampled are illustrated in Figures (2-2) and (2-3). With the switching frequency constant, theoretically only one sample per cycle is needed [Ref. 7] to implement Equation (5-5); however, the noise associated with sampling is significantly reduced by sampling as many times as possible and then averaging the samples over one switching interval. In Chapter III, it was shown that the maximum time between samples for the A/D converters is 2.95 μs. Figure (5-2) shows which A/D converters are used and the physical location on the I/O board used to connect the sensor leads. The following code is the beginning of the PLD B interrupt routine and initializes an A/D conversion:

```
LDI        @inputcs,AR0
LDI        @acs,AR2
```

```
        LDI     *AR0,R0
        LDI     *AR2,R1
        LDI     00CH,R2
wait:   SUBI    01H,R2
        BNZ     wait
```

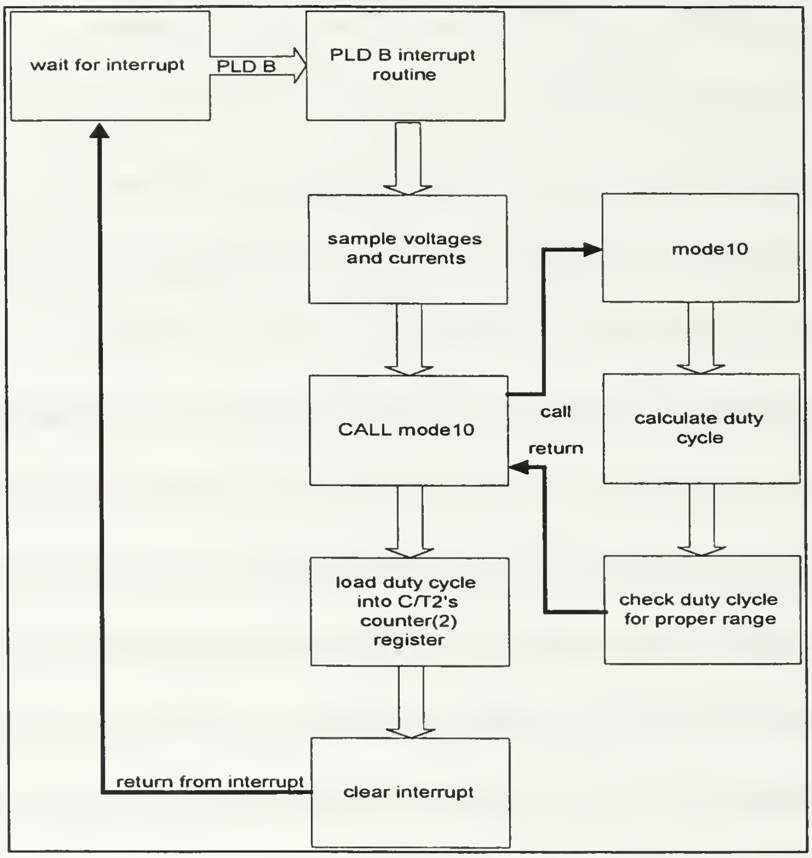

Figure 5-7, Control flow diagram.

The inputcs label is the memory location that contains the address of the converters marked on the I/O board as J7 (input voltage) and J8 (inductor current). The acs label is the memory location that contains the address of the other two A/D converters marked as J1 (output voltage) and J4 (output current). Table (5-2) shows the address and

labels for all ten A/D converters. The first two lines of code loads (load integer) the inputcs address into AR0 and the acs address into AR2. Since the data bus is 32-bits wide and each A/D output is only 12-bits wide, only one address is needed to read two converters.

Label	Address	XD0(LSB) to XD11(MSB)	XD16 (LSB) to XD27 (MSB)
acs	804A00 hex	J1 (output voltage)	J4 (output current)
bcs	804B00 hex	J2	J5
ccs	804C00 hex	J3	J6
inputcs	804900 hex	J7 (input voltage)	J8 (inductor current)
adc1_cs	804D00 hex	J11	-
adc2_cs	804E00 hex	-	J12

Table 5-2, A/D converter addresses and labels.

Recall from Chapter III that the A/D converters are configured such that a LDI instruction with its address as the operand will not only read into the destination register (in this case R1 or R2) the last conversion value but will also initiate a new conversion. The next two lines of code initiate the conversion of the sensed analog signal to a digital signal and reads the previous converted signal. Since the code above initiates the first A/D conversions of the interrupt routine, the value read is the result of an A/D conversion that took place in the previous switching period and therefore, discarded. The conversion process takes 2.95 μs so to get the results of the current conversion, a wait loop that takes ≈ 3.2 μs to run is applied. The wait loop contains the SUBI (subtract integer which takes

50 ns to execute) and the BNZ (branch not zero which takes 0.2 μs to executed) instructions. After the wait loop is completed, another read is initiated but this time the signals obtained are from the current switching period. Table (5-2) lists the bits of the data bus associated with the output of each A/D converter. For example, the instruction

```
LDI        AR0, R0
```

loads the 12-bit digital input voltage into the first 12-bits of register R0 and loads the 12-bit digital inductor current into bits [16,27] of register R0. The voltage and current are separated and stored in separate registers with the following code [Ref. 16]:

```
LDI        R0,R1
LSH        04,R1
ASH        -14h,R1
FLOAT      R1
LSH        14H,R0
ASH        -14H,R0
FLOAT      R0
```

Figure (5-8) illustrates how the above code separates the current and voltage into separate registers and converts them into double precision floating point numbers. After each sample, a wait loop is run to allow another sample to be taken. The maximum number of samples allowed due to the time constraints associated with a 20 kHz switching frequency is 5 samples. This was determined by running the interrupt code with the simulator. After 5 samples are obtained, the samples are averaged and stored into memory.

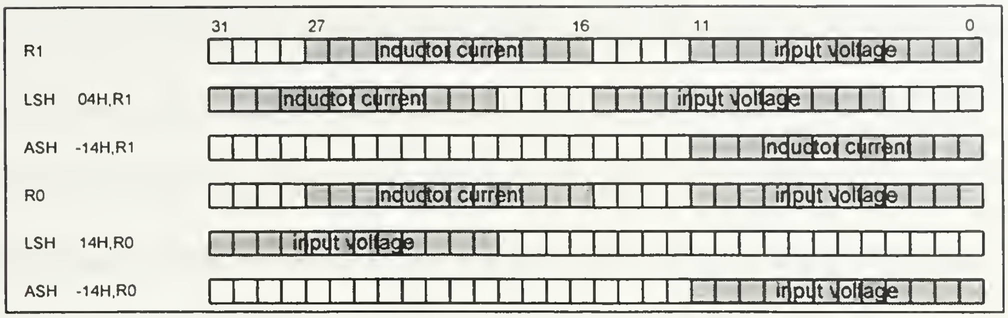

Figure 5-8, Reading the A/D output.

The flow diagram in Figure (5-7) shows that the next task is for the program to call the subroutine mode10. Subroutine mode10 takes the averaged samples obtained above and calculates the duty cycle using Equation (5-5). The code is commented and is shown in Appendix D. After the new duty cycle d[n+1] is calculated, it is loaded into register R7.

As explained earlier, the duty cycle is the number of counts that C/T2 counter(2) contains in its register. The quantization of the duty cycle is defined by the precision of the counter/timers. The smallest amount that the duty cycle can change is by 1 count (100 ns) which corresponds to 1/500 = 0.002 for a 20 kHz switching frequency. There are limits placed on this count number based on the logic program burned into the PLDs. The minimum, in terms of percent of the switching frequency count, is 5% and the maximum is 95%. Therefore, if the calculated duty cycle count is above 95% of tms_swp, R7 is replaced with the maximum value. Likewise if the calculated duty cycle count is below 5% of tms_swp, R7 is replaced with the minimum value.

There is a comparator circuit on the I/O board that that will send a "low" signal to the each PLD if any signal connected to J4, J5, and J6 (from Table (5-2)) exceeds the AC

or DC trip current values entered from the Settings window (see Figure (5-5)). The PLDs could be reprogrammed by NSWC to cause all the switches to open if it received this "low" overcurrent condition. Currently this capability is not programmed in the PLDs.

Once the duty cycle has been determined the subroutine mode10 is completed and the program pointer returns to the PLD B interrupt routine. The contents of register R7 is then stored into the C/T2 counter(2) register. These contents are stored before the next trigger from C/T1 as shown in Figure (5-5). This completes the PLD B interrupt subroutine and the program now waits for the next interrupt.

2. Control of Two Converters

The Universal Controller can control two buck converters with only a slight modification to the code. Figure (5-9) shows which hardware components must be

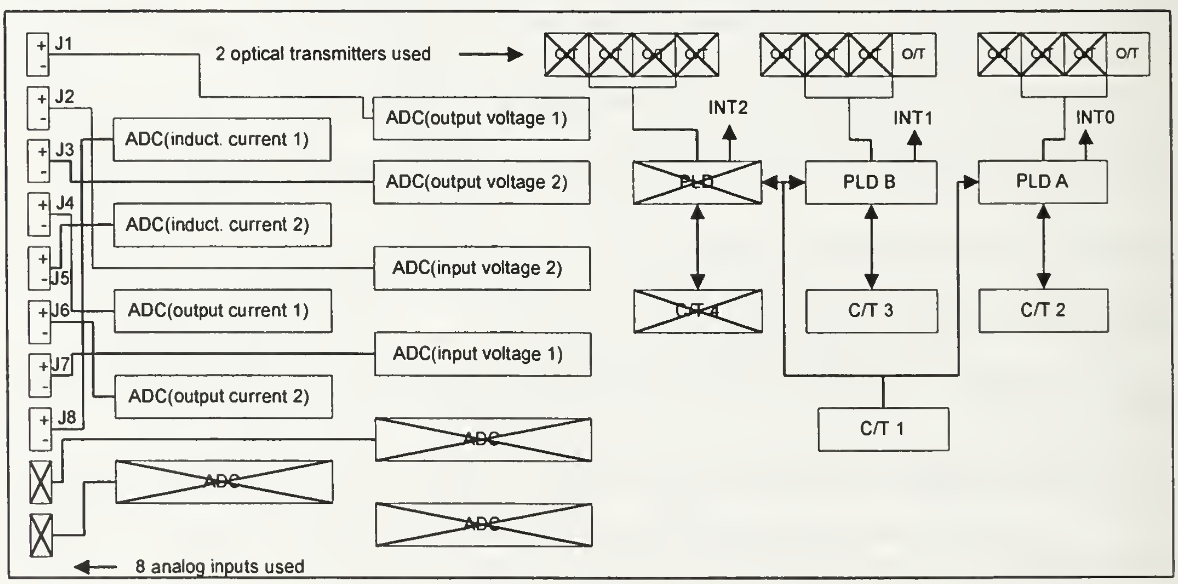

Figure 5-9, I/O board configuration for controlling two buck converters.

configured on the I/O board. The first modification includes enabling both the PLD A and PLD B interrupts. This is done by modifying the last step in the 87C51 interrupt shown in Figure (5-3) by writing 010bH to the Interrupt Enable (IE) register which will enable the interrupts PLD A, PLD B, 87C51, and timer0. Since both buck converters will have the same reference voltage and control gains, no other modifications to this part of the code is necessary.

With both PLD A and PLD B interrupts enabled, Figure (5-7) is modified so that the program flow diagram looks like Figure (5-10). The PLD B interrupt routine still

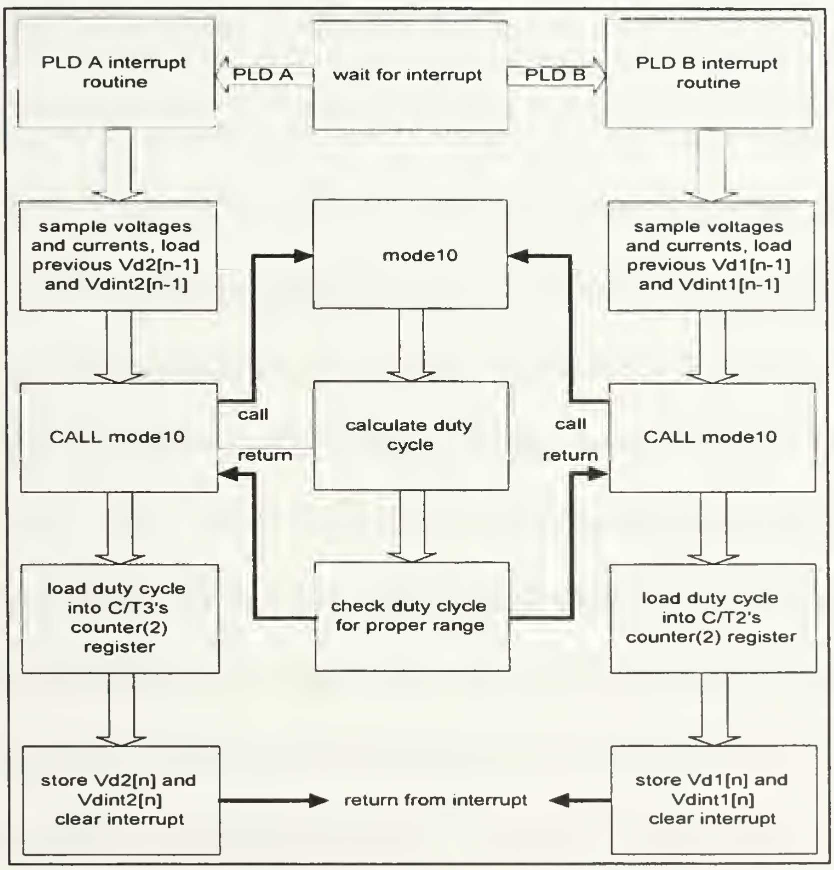

Figure 5-10, Program flow diagram for controlling two buck converters

samples and averages the currents and voltages of the buck converter connected to PLD A (buck 1). Now PLD A is used to sample the voltages and currents of the second buck connected to PLD B (buck 2). Figure (5-4) does not change; therefore, the switching periods will be synchronized in that buck 2 will start its switching period midway into buck 1's switching period. The mode10 subroutine is only slightly modified. Since each buck has a different voltage error (V_d) and integral of the error (V_{dint}), these variables are updated before the mode10 subroutine call shown in Figure (5-10). The mode10 subroutine modifies these variables and uses them to calculate the new duty cycle for each interrupt. After the program returns from the mode10 subroutine, the updated V_d and V_{dint} values are stored in memory to be used for the next duty cycle calculation.

3. Design Development

Much of the design development involved deciphering how the Universal Controller code generated a fixed duty cycle. The task was cumbersome since the only way to test code modifications was to burn the code into the PROMs and then place the PROMs in the controller and check to see if it worked.

The next step was to sample signals using the A/D converters and then convert these digital signals to the proper levels for application in the control algorithm. A simple feed-forward controller was developed to aid in understanding this process. The feed-forward controller simply samples the input voltage and a duty cycle is calculated using only the D_{ss} term (Equation (5-1)).

Probably the most difficult task involved implementing the integrator. Although the trapezoidal integration technique is straightforward, it took many attempts before learning why it was not working properly. The problem was simply a matter of not storing the sampled signals in memory before using them in calculations. If the sampled signals were only loaded into one of the CPU registers and not stored into memory the integrated value became very large. Again this was a long process due to the required PROM burning and testing.

4. Results

The development of the control law relied on digital simulations implemented using the Advanced Continuous Simulation Language (ACSL). Reference [7] contains the complete development of the control law including simulation development and compares the simulated results with the actual results. Figure (5-11) shows the simulation results for a periodic change in load from 10% (291 W) to 100% (2.91 kW) at a frequency of 20 Hz. Both the output voltage (labeled Vout3) and the output current (labeled iout3) are plotted. Figure (5-12) shows the actual hardware results for the same periodic load change with the 3kW buck converter using the Universal Controller. The plots are remarkably similar. The output voltage in Figure (5-12) is AC coupled so that the plot does not show the 208 V DC value that the variations are riding upon. As a result, the noise associated with the sensing devices appears more pronounced in Figure (5-12).

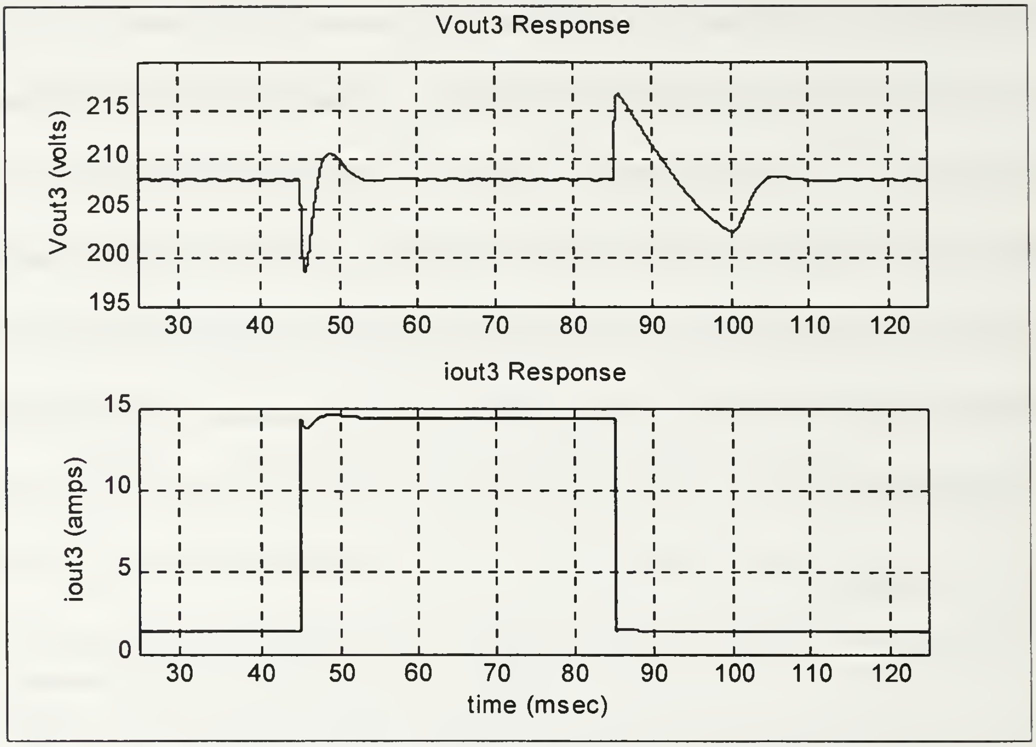

Figure 5-11, Simulation results for 10% to 100% change in load [Ref. 7]

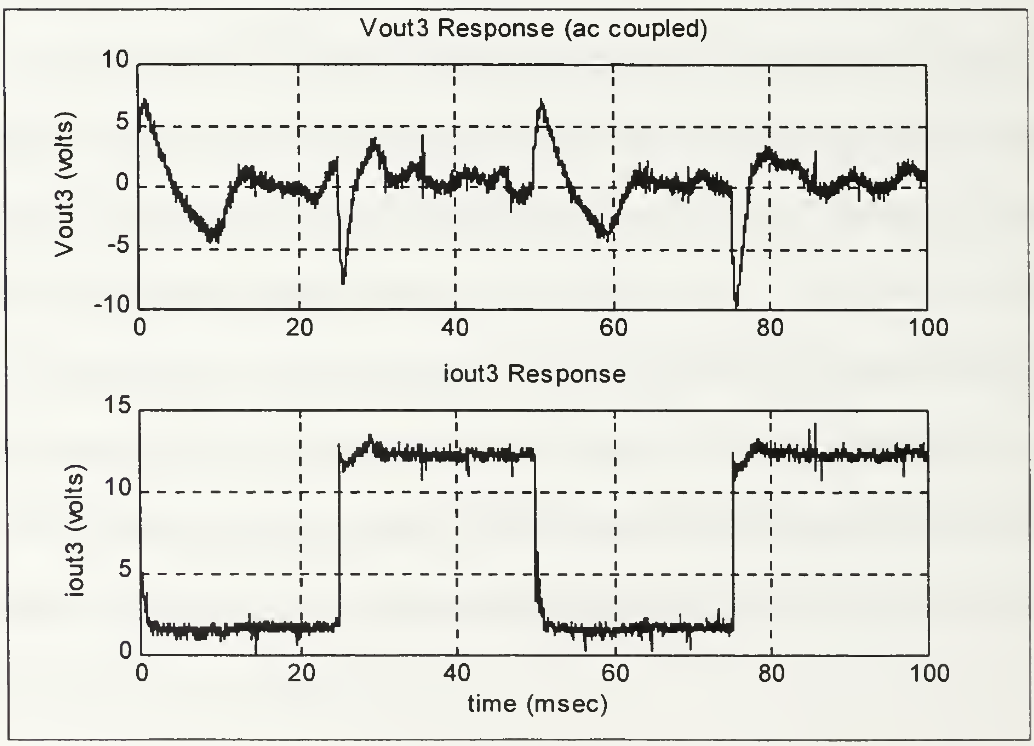

Figure 5-12, Actual results using the buck converter and Universal Controller [Ref. 7].

C. ARCP INVERTER DSP CONTROL IMPLEMENTATION

The Universal Controller was initially designed by personnel at NSWC to control an ARCP. The firmware provided with the controller contained the program to operate the NSWC ARCP inverter both in open and closed-loop configuration. For this reason, most of this research focused on implementing the DC-to-DC converter closed-loop control. In an effort to not reinvent the wheel, the NSWC code was used to test the PENN State ARCP inverter open-loop. As a result of these tests, problems with the ARCP hardware were discovered and consequently this delayed the investigation of the closed-loop control algorithm for the ARCP. The remainder of this chapter contains a discussion of the firmware developed by NSWC for open-loop control of the ARCP inverter and introduces a proposed for a closed-loop implementation.

1. NSWC Open-Loop Implementation

Just as in the case of the buck converter, the I/O board's hardware must be configured properly. Figure (5-13) shows the I/O board hardware that is used to operate the ARCP inverter open loop. All the PLDs and counter timers are used to generate the switching period and duty cycle signals for all three phases of the ARCP inverter. Six optical transmitters are used to send an "upper switch" and "lower switch" gate signal to each phase of the inverter. The NSWC code must be burned into the PROMs and installed on the Universal controller (Appendix C contains the location of the code and instructions on how to burn the PROMs). Then by selecting the "Test mode" under the

Mode menu item on the PC software, the controller will operate the ARCP inverter open loop.

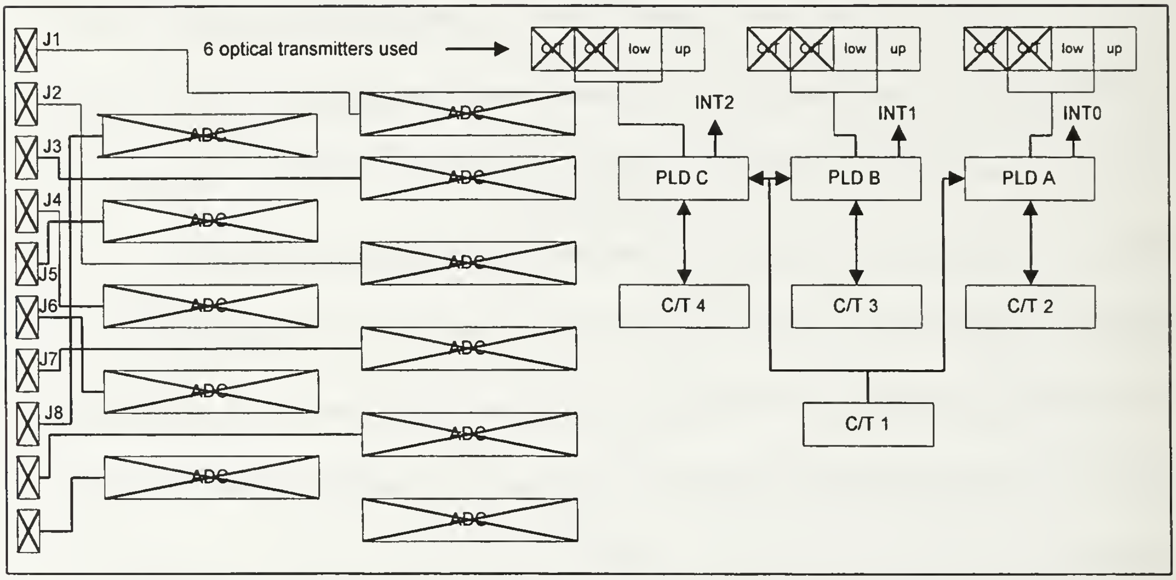

Figure 5-13, I/O board configured to operate the ARCP open loop.

While the "DC-to DC Buck" selection corresponds to cmd10 and mode10, the "Test mode" selection corresponds to cmd1 and mode1. By selecting "Test mode" the PC downloads an integer value of one to memory location 100001h within the dual port memory. The read_cmd subroutine then reads this value and then branches to the portion of code associated with the ARCP open-loop operation. The program flow is documented in Figure (5-14) and is similar to how the Universal Controller is configured for mode10 as shown in Figure (5-3).

The flow diagram shows that there are three subroutines called from the cmd1 section of code. The save_setup subroutine is very similar to the subroutine used in cmd10. It loads all of the parameters entered from the Settings window on the PC to the internal RAM within the C30. Just like the setup for the buck converter, the parameter

tms_swp is calculated and unlike the buck converter the parameter *tms_swp_120* is set to two-thirds of the switching period count. This is used in the init_swct subroutine and will be explained below. It also calculates the following three parameters used in

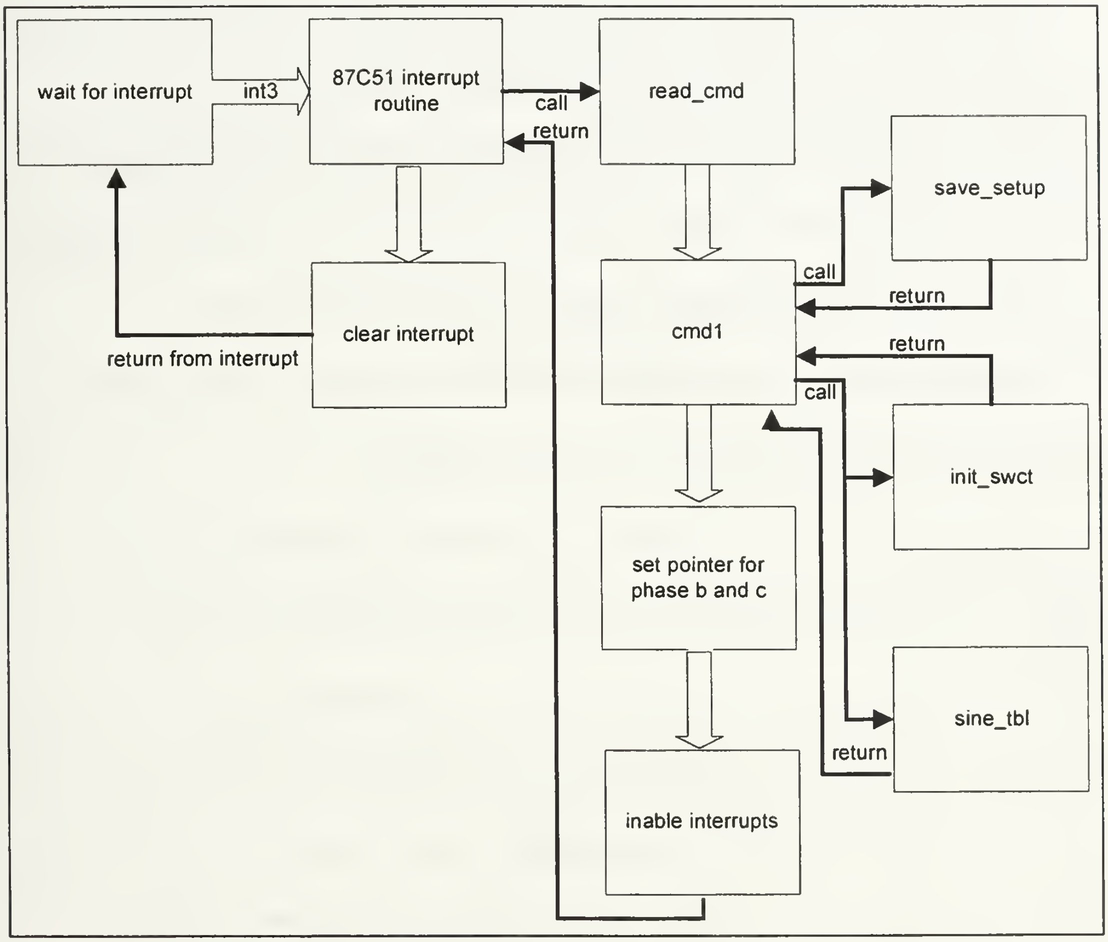

Figure 5-14, Program flow for the open-loop ARCP operation.

generating the PWM signal:

$$tms\_ta = \frac{tms\_swp}{2} \tag{5-9}$$

$$tms\_tb = \frac{tms\_swp - 2(tms\_dt)}{2} \tag{5-10}$$

$$tms\_stepx = \frac{tms\_of * tms\_blk}{tms\_swf} \tag{5-11}$$

where

tms_swp = switching period count calculated in save_setup subroutine

tms_dt = dead time entered from the Settings window

tms_of = desired frequency of the ARCP output entered from the Settings window

tms_blk = block size of the sine-wave table entered from the Settings window

tms_swf = switching frequency entered from the Settings window

The above equations will be explained later in the chapter. For now, it suffices to say that they are calculated at this point in the flow diagram.

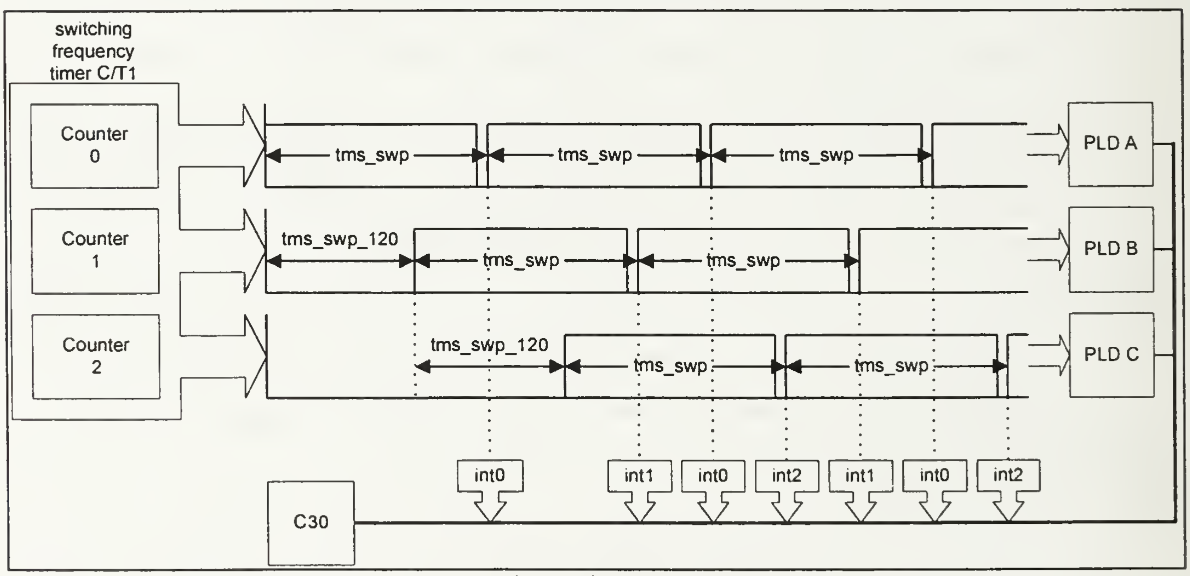

Figure 5-15, Initializing the switching frequency timer counters.

The init_swct subroutine shown in Figure (5-14) is also the same subroutine used for the buck converter except that the value of *tms_swp_120* is different. As explained above, the save_setup subroutine calculated this value to be two-thirds of the switching

period count. So each counter waits this many counts from when the previous counter started before starting its own count as shown in Figure (5-15). This causes each counter output signal to be separated by one-third of a cycle (120 degrees). Thus, this implements the desired phase displacement required in a balanced 3-phase system.

The next task indicated in Figure (5-14) is for the program to call the subroutine sine_tbl. This subroutine simply generates one period of a sine-wave using the following equation:

$$R0 = \sin\left(\frac{2\pi N}{tms\_blk}\right) \tag{5-12}$$

where

N = is an integer value that goes from 0 to (*tms_blk* -1) in unit increments

tms_blk = block size entered from the Settings window on PC

R0 = CPU register 0

Equation (5-12) is implemented in a loop and generates a lookup table which is stored in the SRAM. The length of the lookup table is *tms_blk*.

The last step indicated in the program flow diagram of Figure (5-14) is to set a pointer for phase b and c. The pointer for phase a is the beginning of the sine-wave lookup table. The auxiliary address register 7 (AR7) points to this location. Two other variables *tms_23* and *tms_13* point to an address offset from AR7 by 1/3**tms_blk* and 2/3**tms_blk* respectively. The size of the lookup table is then stored in the block-size register (BK) which is used for circular addressing which will be explained below. Finally, all four interrupts (PLD A, PLD B, PLD C and 87C51) are enabled.

To explain how the PWM signal is generated from the above equations and sine-wave lookup table, it is best to consider only one phase at a time. Figure (5-16) is the flow diagram for phase a. Just like the buck converter, the value stored in the counter(2) register of C/T2 will determine the duty cycle. For the ARCP the duty cycle is calculated using the following equation:

$$dutycount = tms\_tb * R7 + tms\_ta \qquad (5\text{-}13)$$

where

R7 = value read from the sine-wave lookup table

tms_ta = Equation (5-9)

tms_tb = Equation (5-10)

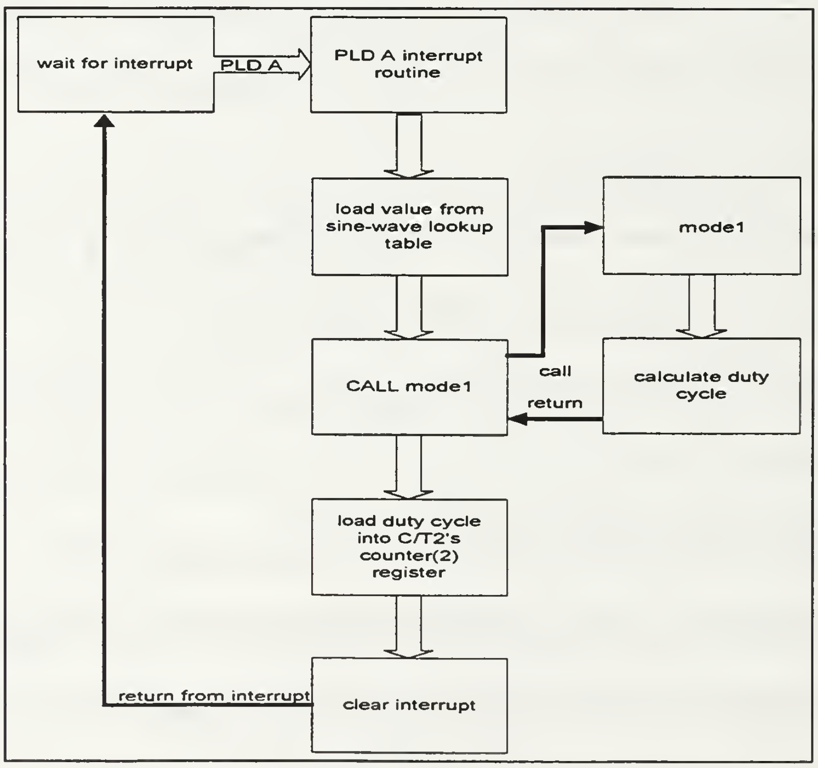

Figure 5-16, PWM signal generation for phase a.

Retrieving the sine-wave value from the table is accomplished using the following code:

```
LDI     *+AR3(tms_stepx), IR1
LDF     *+AR7++(IR1)%,R7
```

This code makes use of the circular addressing capability of the C30 microprocessor. Circular addressing in this case allows the pointer to the lookup table to be incremented by *tms_stepx* (Equation (5-11)) after reading the current memory location pointed to by AR7. Once IR1 reaches the value stored in the block-size register (BK), the pointer moves back to the beginning address.

This code is executed each switching period and for a 20 kHz switching rate that is equivalent to every 50 μs. Equation (5-11) yields a value for *tms_stepx* that will allow the above code to cycle through the lookup table in the desired time corresponding to the output frequency. For example, if the desired output frequency of the ARCP is 60 Hz, this code must complete one circulation of the lookup table in 16.7 ms (1/60 s). If the block size of the lookup table is 2000 and the switching frequency is 10 kHz, *tms_stepx* would be 6 as Equation (5-11) yields. With 6 as the step size, it would take 2000/6= 333 steps to cycle through the lookup table and at 50 μs per step, it would take 333(50 μs) = 16.7 ms for one complete cycle which is the desired period for a 60 Hz output.

The duty count as shown in Equation (5-13) uses Equations (5-9) and (5-10). Figure (5-17) illustrates how the PWM signal is generated. The value of *tms_ta* is set to half of the switching period count, as explained earlier, so that when R7 is zero (at location a in Figure (5-17)) the duty cycle is at 50%. The next time the duty count is

calculated R7 contains the value from the sine-wave lookup table found at location b. Each interrupt the duty count increases until R7 reaches the top of the sine-wave (at

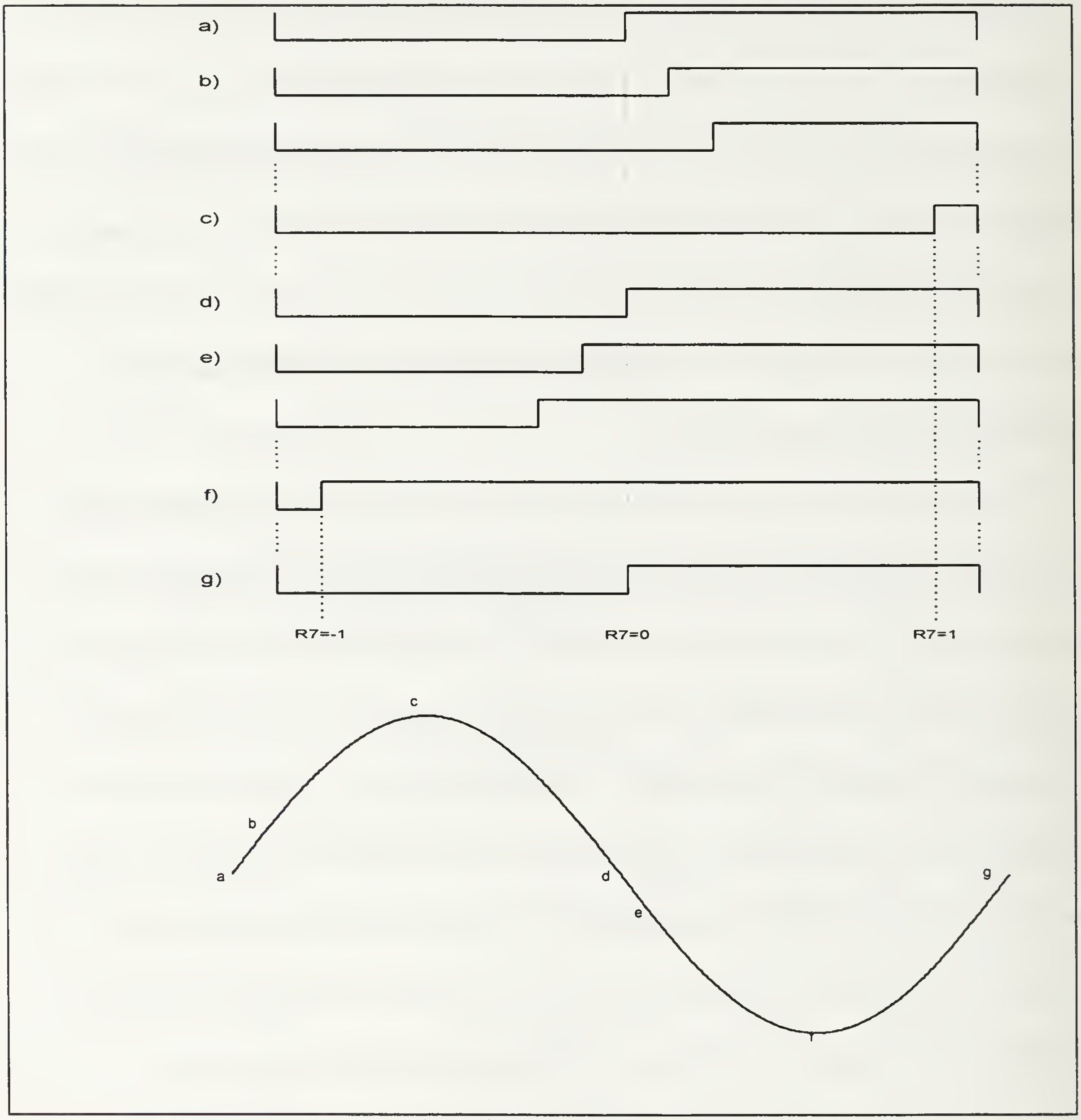

Figure 5-17, Generation of PWM signal using Equation (5-13).

location c). From here the duty count starts to decrease and returns to a 50% duty cycle as shown by location d. The duty count goes below a 50% value once R7 starts to go negative (at location e). The duty count reaches its minimum value when R7 = -1

(location f) and returns back to 50% as shown by location g. The amplitude or maximum duty cycle is controlled by Equation (5-10). The parameter *tms_dt* can be changed from the Settings window on the host PC. By decreasing this value the amplitude of the control sine-wave increases. The minimum *tms_dt* allowed is 14. The reason for this is for the same reason the buck converter duty cycle cannot exceed 95%, the PLD logic will not allow a 100% duty cycle.

2. Proposed Closed-Loop Implementation

The closed-loop control of a three-phase inverter may be implemented in a number of ways including sine-triangle modulation, bang-bang hysteresis control, and space vector modulation [Ref. 10]. Sine-triangle PWM is attractive from the standpoint that the harmonic spectrum is well understood and the switching frequency is fixed by the carrier waveform. The control signals may be established by regulating stationary reference frame quantities or synchronous reference frame quantities. The advantage of manipulating synchronous reference frame quantities is that they are constants in the steady state so that the integral action in a PI controller will generate zero steady-state error. In order to regulate synchronous reference frame quantities, we need to convert the measured abc quantities to the synchronous reference frame by employing a diffeomorphic transformation. At which point, we can apply the PI control algorithm then inverse transform those quantities and retrieve abc control voltages required by the sine-triangle PWM control [Ref. 10].

In this research effort, it was decided that the preliminary closed-loop control for the ARCP would regulate the currents out of each phase. Currents were selected as opposed to voltages since the ARCPs were already equipped with sensors providing scaled measurements of the currents. For the following algorithm description, superscript 's' will denote the stationary reference frame, superscript 'e' will denote the synchronous reference frame, subscripts 'abc' will denote actual phase quantities, subscripts 'qd' will denote transformed quantities, and a superscript '*' will denote a commanded value.

The algorithm is initiated by specifying the commanded synchronous reference frame currents:

$$i_q^{e*} = \text{amplitude of desired control} \tag{5-14}$$

$$i_d^{e*} = 0 \tag{5-15}$$

Here the d-axis commanded current is arbitrarily set to zero for convenience. If the load is a three-phase induction machine, then this commanded value may be fixed at some other number. To calculate the stationary reference frame currents, two of the three phase currents must be sampled and transformed using the following transformation:

$$\begin{bmatrix} i_q^s[\mathrm{n}] \\ i_d^s[\mathrm{n}] \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{\sqrt{3}}{3} & -\frac{2\sqrt{3}}{3} \end{bmatrix} \begin{bmatrix} i_a[\mathrm{n}] \\ i_b[\mathrm{n}] \end{bmatrix} \tag{5-16}$$

where

$i_a[n]$ = sampled phase a current

$i_b[n]$ = sampled phase b current

The above equation assumes that the phase currents necessarily sum to zero. The next task is to transform the stationary reference frame currents to synchronous reference frame currents using the following transformation:

$$\begin{bmatrix} i_q^e[n] \\ i_d^e[n] \end{bmatrix} = \begin{bmatrix} \cos(\theta_e[n]) & -\sin(\theta_e[n]) \\ \sin(\theta_e[n]) & \cos(\theta_e[n]) \end{bmatrix} \begin{bmatrix} i_q^s[n] \\ i_d^s[n] \end{bmatrix} \tag{5-17}$$

where

θ_e = electrical angle of the desired abc quantities

The values of $\sin(\theta_e)$ and $\cos(\theta_e)$ can be found using the sinewave lookup table in the same manner as in the open-loop case described above. The PI control signals are generated using the commanded synchronous reference frame currents in Equations (5-14) and (5-15) and the calculated synchronous reference frame currents in Equation (5-17). Using the discrete integrator implementation of Equation (5-4) the PI control equations becomes:

$$V_{q,PI}^e[n] = K_{pq}\left(i_{d,q}[n]\right) + K_{iq}\left\{ i_{dint,q}[n-1] + \frac{T}{2}\left(i_{d,q}[n-1] + i_{d,q}[n]\right)\right\} \tag{5-18}$$

$$V_{q,PI}^e[n] = K_{pd}\left(i_{d,d}[n]\right) + K_{id}\left\{ i_{dint,d}[n-1] + \frac{T}{2}\left(i_{d,d}[n-1] + i_{d,d}[n]\right)\right\} \tag{5-19}$$

where

$i_{d,q} = i_q^{e*}[n] - i_q^e[n]$

$i_{d,d} = i_d^{e*}[n] - i_d^e[n]$

K_{pq} and K_{pd} = proportional gains

K_{iq} and K_{id} = integral gains

$$i_{\text{dint,q}}[\text{n}] = i_{\text{dint,q}}[\text{n}-1] + \frac{\text{T}}{2}\left(i_{\text{d,q}}[\text{n}-1] + i_{\text{d,q}}[\text{n}]\right)$$

$$i_{\text{dint,d}}[\text{n}] = i_{\text{dint,d}}[\text{n}-1] + \frac{\text{T}}{2}\left(i_{\text{d,d}}[\text{n}-1] + i_{\text{d,d}}[\text{n}]\right)$$

Before abc-control signals are calculated, the control signals above must be inverse transformed back to the stationary reference frame using the following inverse transformation:

$$\begin{bmatrix} \text{V}^{s}_{q,PI}[\text{n}] \\ \text{V}^{s}_{d,PI}[\text{n}] \end{bmatrix} = \begin{bmatrix} \cos(\theta_e[\text{n}]) & \sin(\theta_e[\text{n}]) \\ -\sin(\theta_e[\text{n}]) & \cos(\theta_e[\text{n}]) \end{bmatrix} \begin{bmatrix} \text{V}^{e}_{q,PI}[\text{n}] \\ \text{V}^{e}_{d,PI}[\text{n}] \end{bmatrix} \tag{5-20}$$

The stationary reference frame voltages are then used to calculate the abc-control signals:

$$\text{V}_{\text{a,PI}} = \text{V}^{\text{s}}_{\text{q,PI}} \tag{5-21}$$

$$\text{V}_{\text{b,PI}} = -\frac{1}{2}\text{V}^{\text{s}}_{\text{q,PI}} - \frac{\sqrt{3}}{2}\text{V}^{s}_{\text{d,PI}} \tag{5-22}$$

$$\text{V}_{\text{c,PI}} = -\text{V}_{\text{a,PI}} - \text{V}_{\text{b,PI}} \tag{5-23}$$

These control signals are translated to dutycount for each phase as explained in the previous section using Equation (5-13). The following dutycounts are then written to the appropriate counters:

$$dutycount_a = \text{V}_{\text{a,PI}} * R7 + tms\_ta \tag{5-24}$$

$$dutycount_b = \text{V}_{\text{b,PI}} * R7 + tms\_ta \tag{5-25}$$

$$dutycount_c = \text{V}_{\text{c,PI}} * R7 + tms\_ta \tag{5-26}$$

where

R7 is a CPU register

tms_ta = 50% of switching period count.

The addresses for the phase counter/timers are shown in Figure (3-3) (ct_phasea, ct_phaseb, and ct_phasec). The following code illustrates how the phase a counter/timer is updated with the new dutycount in register R7:

```
LDI   @ct_phasea,AR0
STI   R7,*+AR0(2)        ; Store LSB of counter 2
LSH   -08H,R7
STI   R7,*+AR0(2)        ; Store MSB of counter 2
```

The steps needed to go from open-loop control to closed-loop control are simply software related. The PI gains and the desired synchronous reference frame commanded current values can be entered from the Settings window. It is anticipated that once this algorithm is successfully implemented, extensions to more exotic algorithms and electric machine controls will be quite straight forward.

VI. CONCLUSIONS

A. SUMMARY OF RESEARCH WORK

The Universal Controller was designed by engineers at NSWC and populated, soldered and configured at NPS by the Power Systems Group. The DSP controller allows for greater flexibility in control implementation which enhances the research efforts pertaining to the DCZEDS program. The focus of this research has been to document how the Universal Controller works and then apply this knowledge to implementing closed-loop control algorithms for a buck converter and a 3-phase inverter.

In Chapter II, the basic operation and specifications for the 3 kW DC-to-DC converter (SSCM) and the 30 kW ARCP Inverter (SSIM) were discussed, and the motivation for using DSP control was addressed. There are many commercial boards that are available here at NPS, and in Chapter III an overview of these boards was presented along with the specific limitations that made them unsuitable for application in SSCM and SSIM control. Next, the hardware that makes up the Universal Controller was described and investigated. In Chapter IV the firmware for initializing and interfacing all of the hardware components was presented along with the basic operation of the primary components. The PC software which interfaces with the Universal Controller was also described. The implementation of the closed-loop control for the DC-to-DC converter was described and documented in Chapter V. Chapter V also contains an explanation and supporting documentation for the open-loop control of the ARCP inverter. Finally, an

outline of the modifications required to implement closed-loop control for the ARCP inverter was presented.

B. NOTABLE CONCLUSIONS

The Universal Controller is a powerful and useful tool. The hardware is uniquely setup to facilitate controlling not only the SSIM and the SSCM but many other PEBB applications. The closed-loop control implementation for the DC-to-DC converter yielded acceptable results that mirrored those obtained for simulation.

There were a number of limitations discovered while analyzing and programming the Universal Controller. One of these limitations includes the inability to program a 100% duty cycle. While this was not a problem for the tests that were conducted in Reference [4], it can limit the dynamic range of the linear control and introduce additional harmonics. Another limitation that can be resolved by changing the logic within the PLDs is to enable the PLD to shutdown when the overcurrent interrupt occurs.

C. RECOMMENDATIONS FOR FUTURE WORK

Currently all the programming for the Universal Controller is done using TMS320 assembly language. The program is then burned into the PROM before installing it on the controller. This increases the development time to test and debug a new program. Writing the code in a higher level language like C would make it easier to develop and debug new or modified control algorithms. Another time saver would be to modify the host PC software to allow the user to download a program or subroutine into memory without reburning the PROM. The Settings window should be modified or a new

Settings window for the buck converter added to facilitate the variables unique to buck converter control. Other recommendations include:

- developing the ARCP current control and extending it to voltage control
- running multiple Universal Controllers from one PC
- investigating other buck control algorithms
- exploring different implementations for the discrete integration
- investigating digital filtering of signals to reduce the network noise.

The Universal Controller is an integral part of the PEBB vision. The advantages of the Universal Controller over an analog controller for this application are numerous. The added flexibility that this controller adds to the DCZEDS network simulator makes this a valuable research tool.

APPENDIX A. PARTS LISTING FOR THE UNIVERSAL CONTROLLER

Part #	QTY IO	QTY CPU	Part Description
C0110 95k11	0	1	16 MHz Clock
C0110 95k08	0	1	40 MHz Clock
VF150 8840	1	0	20 MHz Clock
IDT7130 SA35P	0	1	1kx8 dual-port RAM
D87C51FA	0	1	microprocessor
DM74AS373N-ND	1	1	20-dip octal latch w/ tri-state
CY7C274-30WC	0	4	32kx8 EPROM
MC74HC14AN	0	1	14-dip Inverting Schmitt Trigger
MM74HC08N-ND	0	1	14-dip quad 2-input AND gate
TMS320C30GEL40	0	1	DSP chip
IDT71256SA35P	0	4	SRAM
DM74AS244N-ND	0	1	20-dip octal tri-state buffer
SN74LS244N	4	0	Octal buffers
MM74HC244N-ND	2	0	Octal buffers
SN74HC244N	2	0	Octal buffers
EPM5016 DC-15	0	1	Programmable Logic
HP9429 T1521	12	2	Opto Transmitter
HP9507 R2521	0	2	Opto Receiver

Part #	QTY IO	QTY CPU	Part Description
AD536AJD	6	0	RMS to DC Converter
LF347BN-ND	4	0	14-dip quad JFET op amp
HA-1-4900-8	2	0	Precision quad Comparator
LF13508D-ND	1	0	16-dip 8-Channel JFET Analog Multiplexer
MAX120CNG-ND	11	0	12-bit AD Converter
AD7247JN	1	0	AD Converter
CD74HCT377E	5	0	Octal D-Type F-F w/ Data Enable
DM7427N-ND	1	0	14-dip triple-input NOR gate
CD74HCT32E	2	0	14-dip quad 2-input OR gate
DM74AS04N-ND	1	0	14-dip hex inverter
CD74HCT138E	1	0	16-dip 3-to-8 line decoder
CP82C54-12	4	0	Programmable Interval Timer
CD74HCT154E	1	0	24-dip 4-to-16 line decoder
EPM5032 DC-25	3	0	Programmable Logic (?)
SN74LS93N	1	0	4-bit Binary Ripple Counter
96 sock conn	0	1	3-row 96-pin straight connector
96 plug conn	1	0	ditto
12-pin jump	0	1	12-pin jumper
2-pin jump	0	2	2-pin jumper

Part #	QTY IO	QTY CPU	Part Description
16-pin strai conn	1	0	same
10-pin strai conn	1	0	same
34-pin strai conn	1	0	same
5pos DIP sw	1	0	same
SW403-ND sw	0	1	Momentary Switch
p4887-ND .1uF	30	14	same
47uF (50V)	2	0	same
p1259-ND 100uF	2	0	same
p1255-ND 10uF	0	1	same
p2101-ND 22uF	11	0	same
p2063-ND 4.7uF	6	0	same
22k 8-pin Res	0	3	same
2.2k 10-pin	1	0	same
330 16-pin	2	0	same
82k resistor	0	1	same
15k pot resis	1	0	same
10k resistor	4	0	same
2.2k resistor	1	1	same
330 resistor	0	2	same
270 resistor	0	2	same

Part #	QTY IO	QTY CPU	Part Description
Green LED	0	1	same
Red LED	0	1	same
6lyr IOBRD	1	0	same
CPU BRD	0	1	same
28pin wd sock	0	8	same
28pin nar sock	3	0	same
14pin sock	0	1	same
20pin nar sock	0	1	same
40pin wd sock	0	1	same
48pin wd sock	0	1	same
181pin sock	0	1	same
Terminal Blocks	12	1	same
5m HFBR-PWS005	12	4	Fiber-Optic Cabling

APPENDIX B. SCHEMATICS FOR THE UNIVERSAL CONTROLLER

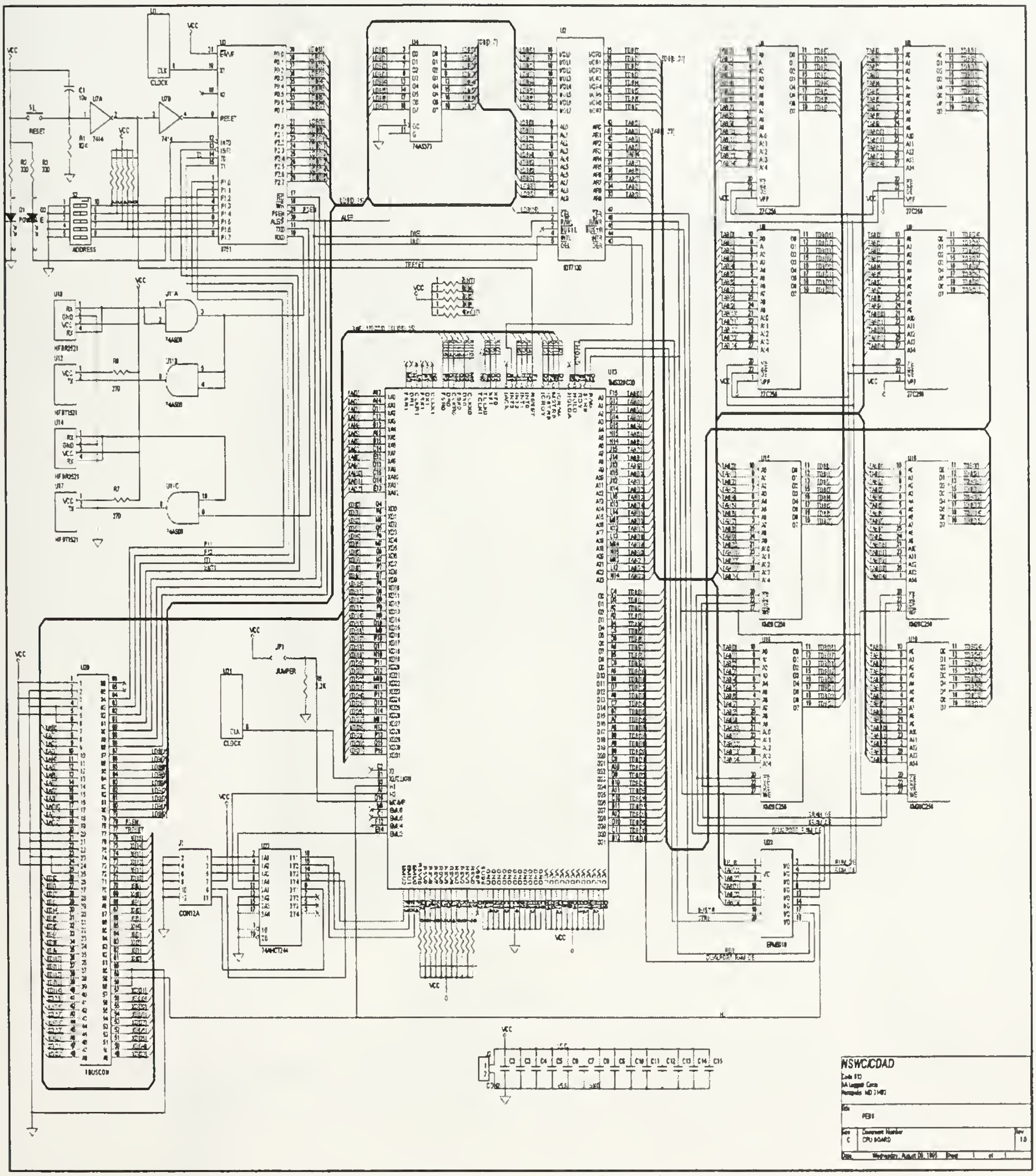

Figure B-1 - CPU board.

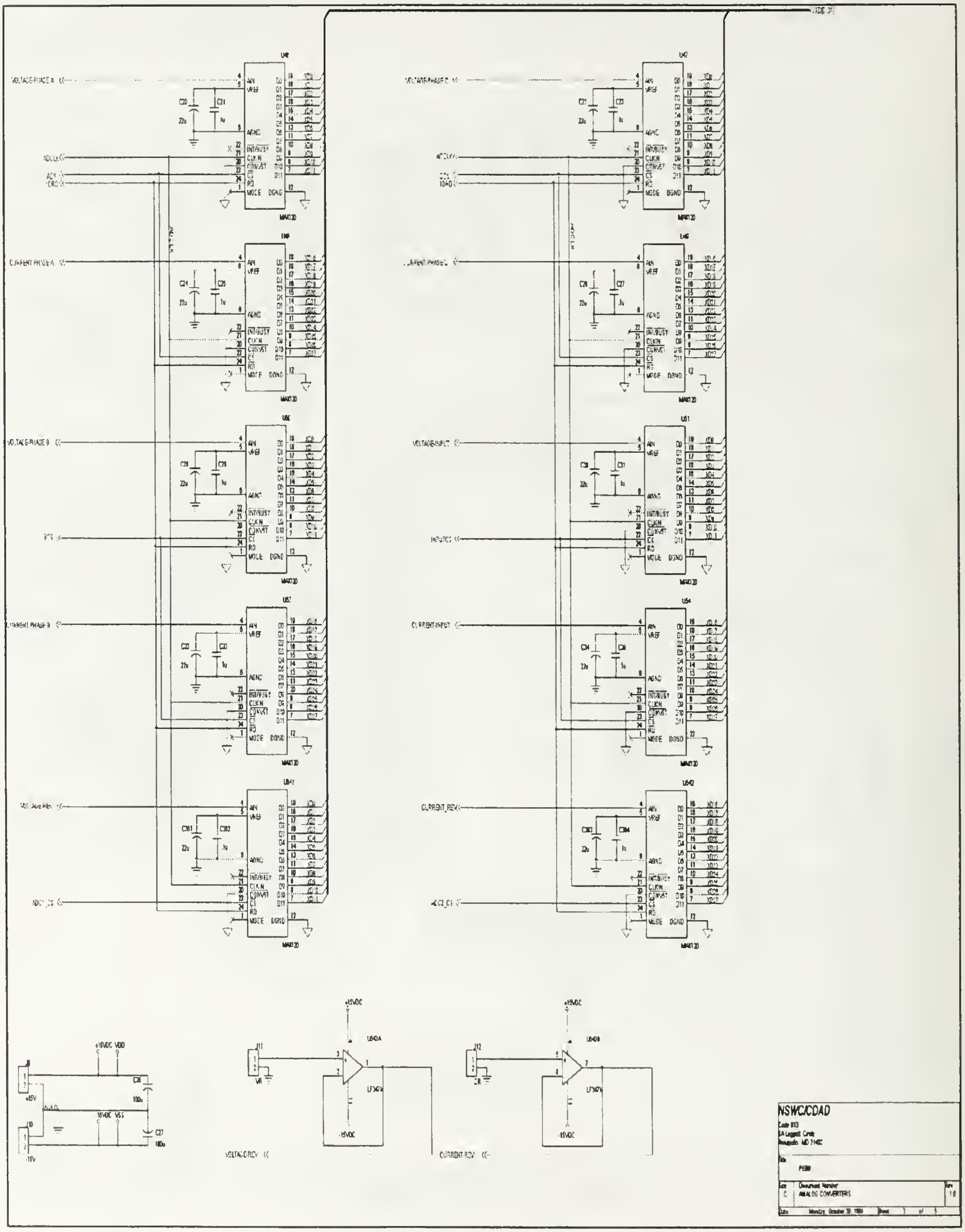

Figure B-2 - Analog converters.

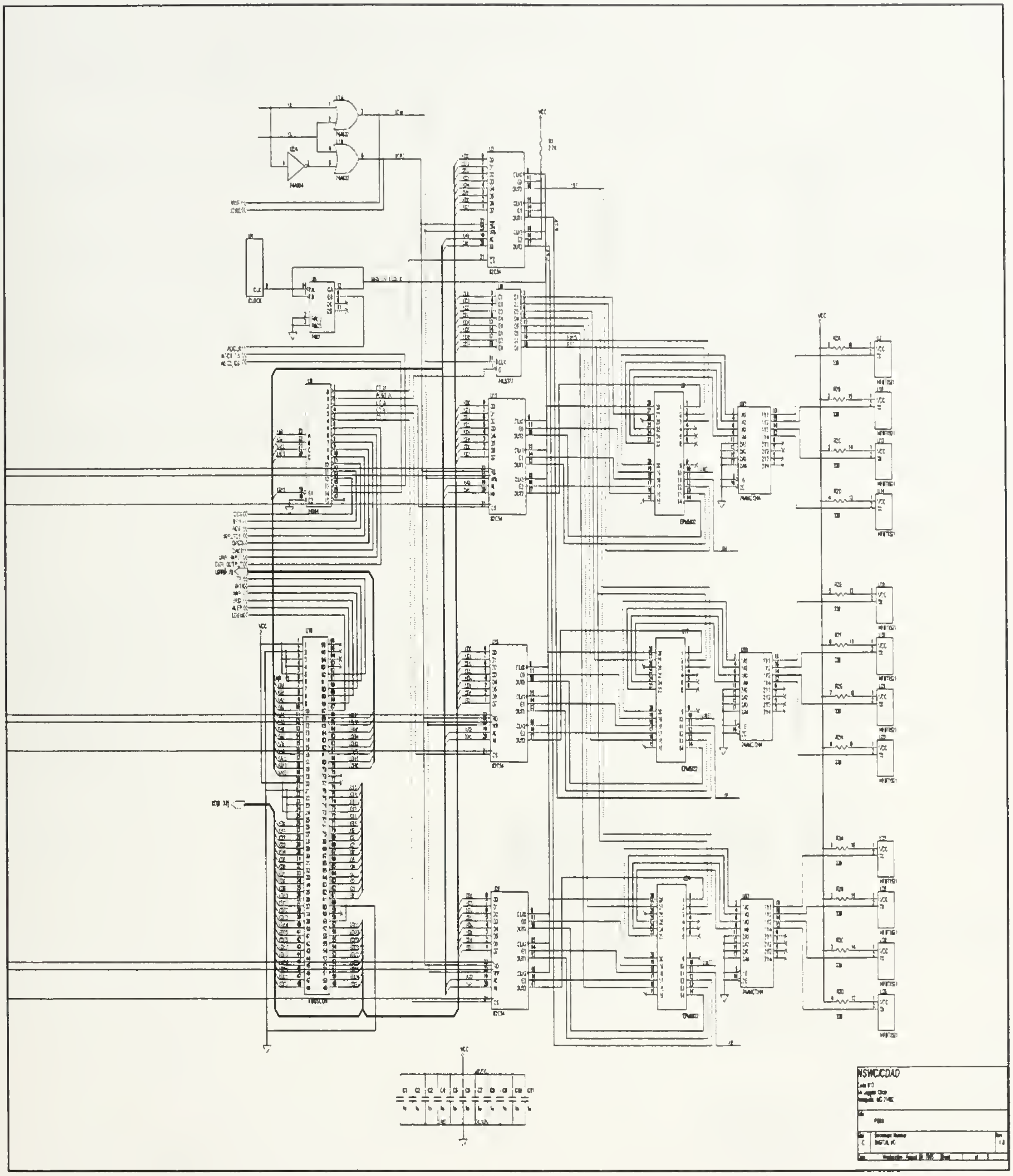

Figure B-3 - The counter/timers and optical transmitters.

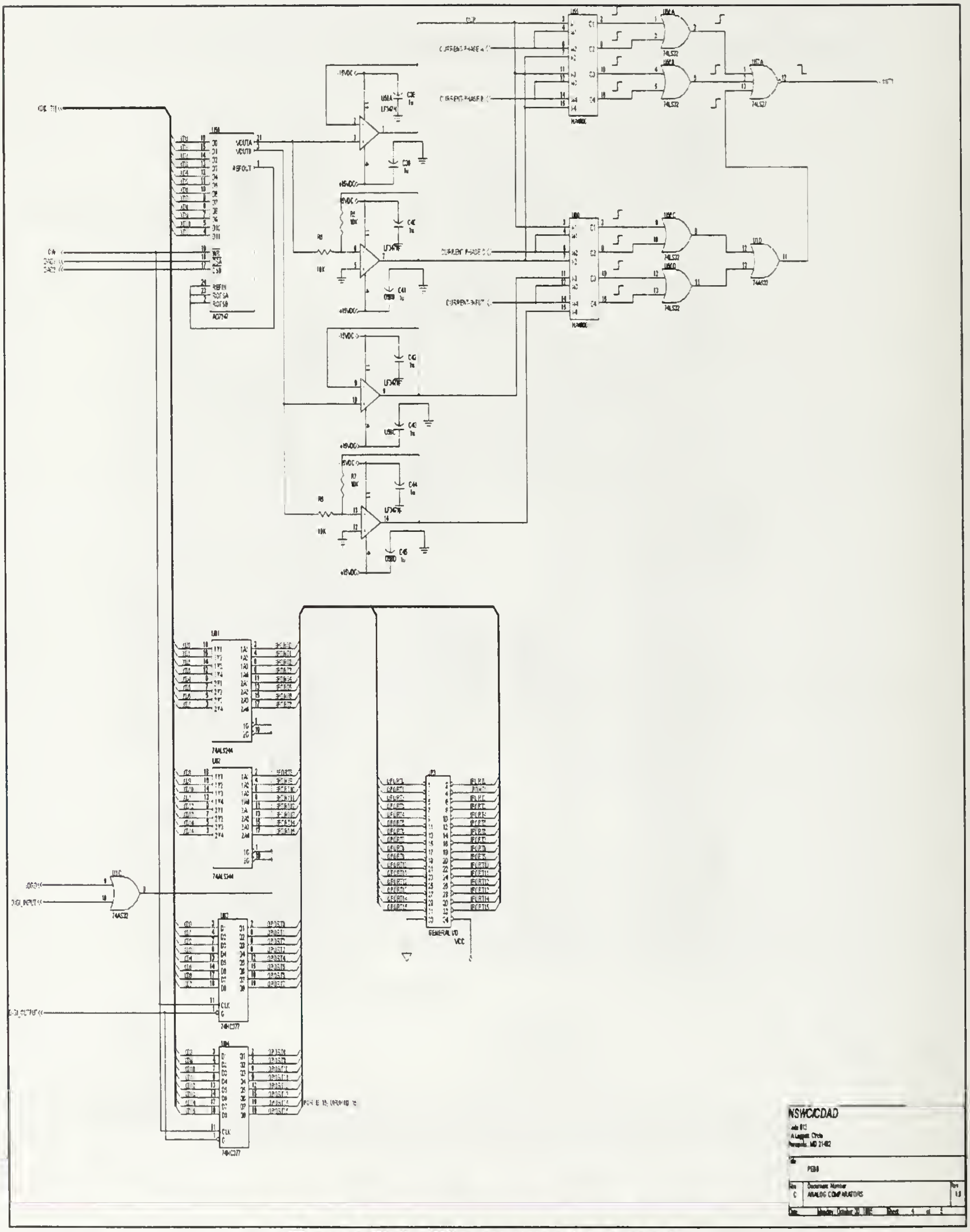

Figure B-4 - The comparator circuit.

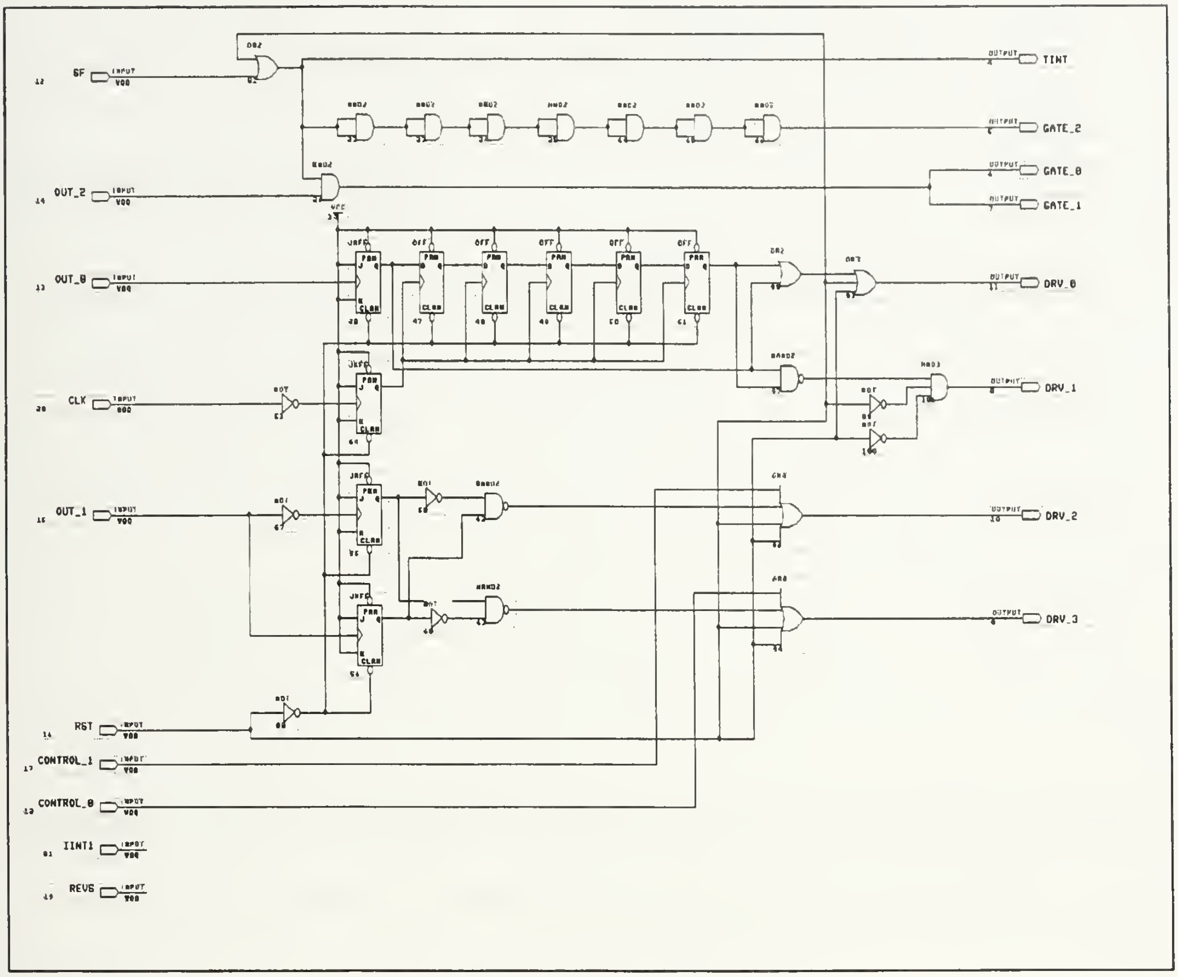

Figure B-5 - The PLD logic.

APPENDIX C. SOFTWARE INSTALLATION

A. HOST PC SOFTWARE

The software is located on PCPWR7 in the Power Systems group laboratory. There are five files located in the c:\PebbUC\hostpc directory. The file names are:

- Pebb.exe
- Cmdialog.vbx
- Mscomm.vbx
- Spin.vbx
- Threed.vbx

To install the software on a Window 3.1 machine (this will not work in Windows 95) just load all the vbx files under the c:\windows\system directory and load the Pebb.exe in any directory.

The software is installed on the only Windows 3.1 machine located in the Power Systems Group Laboratory #m051992. The Pebb.exe is installed in the c:\PEBB\win31new directory.

The C++ code that generated the Pebb.exe file above is located on the PCPWR7 machine in the c:\PebbUC\hostpc directory and is saved as a C++ project in compressed format. The two files Pebb.zip and Pebbres.zip are written in Visual C++ version 1.5 and will not work with version 4.0 (Windows 95).

B. INSTALLING CODE ON THE EPROMS

There are two different assembly codes that are mentioned in this thesis. The code received from NSWC for the ARCP (Pebbfix.asm) and the code developed for the buck chopper (npsbuck.asm). Both of these codes are stored on PCPWR7 in the c:\PebbUC\c30 directory.

The assembler is located on the DOS PC #M051273 under the c:\DSPTOOLS directory. There are many files in this directory and among them are two files that have been setup to aid in the assembly process. The first file is a batch file called "NPSBUCK.BAT". This file requires the input file name (the file you want to assemble) to be npsbuck.asm and that it resides in the same directory it is in. By typing "npsbuck" at the command prompt the batch file will do the following:

- assemble the npsbuck.asm file
- link the output object file produced from the assembler with a command file
- convert the linked file into Intel hex for downloading to the EPROM

The second file is "NPSBUCK.CMD" and this is the file that the linker uses in the batch file described above. The contents of this file are as follows:

```
math.obj divi.obj
-heap 0x800
-m npsbuck.map
-e init
-o npsbuck.out
MEMORY
{
    VECS:        org = 00000h     length = 0040h
    ROM:         org = 00040h     length = 01fc0h
    SRAM:        org = 080000h    length = 080000h
    DUAL_PORT:   org = 100000h    length = 080000h
    XBUS:        org = 804000h    length = 2000h
```

```
    RAM1:          org = 809800h        length = 0400h
    RAM2:          org = 809c00h        length = 0400h
}

SECTIONS
{
    vecs:      load = VECS
    .text:     load = ROM
    .data:     load = ROM
    sram:      load = SRAM
    dualport:  load = DUAL_PORT
    xbus:      load = XBUS
    ram1:      load = RAM1
    ram2:      load = RAM2
}
```

The math.obj and divi.obj are subroutines used by the program. The subroutine math.obj is provided by Texas Instruments and contains many math functions like the sine function use to produce the sinewave lookup table. The divi.obj is a subroutine written by an engineer at NSWC to divide integers.

The -heap 0x800 is used to set the heap size for the C memory pool command malloc() to 800 words. The -m, -e, and -o are linker options that produce a map listing of the output, defines a global symbol that specifies the primary entry point for the output module, and specifies the name of the output file, respectively.

The MEMORY and SECTIONS headings are linker directives. The MEMORY directive allows the memory to be configured as shown. And the SECTIONS directive controls how sections are built and allocated.

The output will consist of four Intel hex files NPSBUCK.I0, NPSBUCK.I1, NPSBUCK.I2, and NPSBUCK.I3. These are the files that must be downloaded to the four EPROM chips. A batch file named CDU.BAT is located in the same directory as the

other files mentioned above. By typing cdu at the prompt, the Universal Programmer and Tester (EPROM burner) will open. The Device menu item will be highlighted. Press the D key and a drop down menu will be shown. Select EPROM from this dropdown menu and a series of windows will ask what manufacturer and type of EPROM you have. Once you selected the desired EPROM, you are now ready to load the hex file to the buffer. To do this select 2 (Load Bin or Hex file to buffer) from the selection window. The window will now giver you the directory listing. Find the c:\DSPTOOLS\NPSBUCK.I0 file and highlight it and press enter. The program will then prompt you for file type, enter “I” for Intel hex. Next it will ask you “file start seg. (0000):” just hit the enter key. The last prompt will ask what you want the unused bytes to be. Enter “1” for zeros, and then hit the enter key. Now the file NPSBUCK.I0 is loaded into the buffer. Place a blank EPROM in the socket on the programmer module and enter “A”. This will load the program into the EPROM. When it is done loading it will beep and then you should repeat the same process over until all four EPROMS are loaded. The EPROM with NPSBUCK.IO is placed in socket U5 on the CPU board; likewise *.I1 goes in U6, *.I2 goes in U8, and *.I3 goes in U9.

C. MICROCONTROLLER SOFTWARE

The assembly code for the microcontroller is located in the PCPWR7 computer in the c:\PebUC\87C51 directory. The assembler for this device is on the same machine as the host PC software in the power lab. The directory, c:\Pebb\intel\compiler, contains the assembler and the object code (.hex extension) in the format accepted by the EPROM

programmer. This file must be copied to a floppy disk and then transferred to the DOS computer to be loaded into the buffer of the programmer in the same manner as above except that the device picked must be the 87C51.

D. PROGRAMMABLE LOGIC DEVICES (PLDs)

There are two PLD programs that are required by the Universal Controller. One for the three PLDs that generate the duty cycle and the other for the address decoder. The PLD code is located in the c:\PebbUC\PLDs directory. The files have the "pof" extension which means they are in the format needed by the EPROM programmer. The duty cycle PLD code is in the dcpld.pof file and the decoder PDL code is in the decoder.pof file. To program this into the chips, copy these files on a floppy disk and transfer them to the DOS computer to be loaded in the program buffer of the programmer.

APPENDIX D. ASSEMBLY CODE FOR DC-TO-DC CONVERTER

```
*****************************************************************
*
*                     NPS POWER LAB
*          TMS320C30 MODIFIED CONTROL CODE
*                     BY RON HANSON
*                       25 Feb 97
*
*****************************************************************
*
         .title    "BUCK"
*
         .global reset,init
         .global int0,int1,int2,int3
         .global tint0
         .global isr0,isr1,isr2,isr3
         .global time0
         .global FPINV,divi
*
*
         .sect    "vecs"                    ; Named section
reset    .word    init            ; RS- loads address init to PC
int0     .word    isr0            ; INT0- loads address int0 to PC
int1     .word    isr1            ; INT1- loads address int1 to PC
         .space 1                                 ;
int3     .word    isr3            ; INT3- loads address int3 to PC
*
         .space    4              ; Reserved space
tint0    .word    time0           ; Timer 0 interrupt processing
         .space    34             ; Reserved space

*
         .data
sram     .word    0080000H        ; Beginning address of SRAM
blk0     .word    0809800H        ; Beginning address of RAM block 0
blk1     .word    0809C00H        ; Beginning address of RAM block 1
stck     .word    0809F00H        ; Beginning of stack
ctrl     .word    0808000H        ; Pointer for peripheral-bus memory map
xbus     .word    0000048H        ; Xpansion bus: 2 wait states, external
*                                 ; RDY not in use (88)
pbus     .word    0000428H         ; Primary bus : 1M bank compare, 1 wait
```

```
*                               ; states, external RDY not in use
tim0ctl       .word  00003C1H   ; Internal timer 0: 1111000001; (301) 1100000001
*
*
*
dp_mem        .word  0100000H   ; Pointer for dual port memory (command reg)
dp_int        .word  00003FEH   ; Pointer for setting interrupt flag
dp_cint       .word  00003FFH   ; Pointer for clearing interrupt flag
*
tms_oaci      .set   0000001H   ; Ac trip current level
tms_acv       .set   0000002H   ; test
tms_bdly      .set   0000003H   ; Boost delay
tms_btime     .set   0000004H   ; Boost time
tms_odci      .set   0000006h   ; Dc trip current level
tms_Vref      .set   0000007H   ; Dc voltage
tms_dt        .set   0000008H   ; Deadtime
tms_of        .set   0000009H   ; Ac frequency
tms_swf       .set   000000aH   ; Switching frequency
tms_blk       .set   000000cH   ; Block size
tms_acs       .set   000000dH   ; current sensor
tms_dcs       .set   000000eH   ; voltage sensor
tms_step      .set   000000fH   ; Step
tms_delay     .set   0000010H   ; Delay
tms_swp       .set   0000011H   ; Switching period
tms_stepx     .set   0000012H   ; Step
tms_ta        .set   0000013H   ; ta constant
tms_tb        .set   0000014H   ; tb constant
tms_kc        .set   0000015H   ; Control constant
tms_kcb       .set   0000016H   ; Control constant
tms_bt        .set   0000017H   ; Control constant
tms_bi        .set   0000018H   ; Control constant
tms_mode      .set   0000019H   ; Mode
tms_swp_120   .set   000001aH   ;
UMIN          .set   000001eH   ; Minimum duty cycle
UMAX          .set   000001fH   ; Maximum duty cycle
Vdiff         .set   0000020H   ; Vout-Vref
Vd_int        .set   0000021H   ; integral of Vdiff
hn            .set   0000022H   ; integrator gain
hv            .set   0000023H   ; voltage gain
hi            .set   0000024H   ; current gain
iL            .set   0000025H   ; sensed inductor current
iout          .set   0000026H   ; sensed output current
Vdiffa        .set   0000027H   ; converter a Vdiff
Vdiffb        .set   0000028H   ; converter b Vdiff
```

```
Vd_inta       .set    0000029H      ; converter a Vd_int
Vd_intb       .set    000002aH      ; converter b Vd_int
Vout          .set    000002fH      ; converter output
Vin_inv       .set    0000030H      ; inverse of converter output
vperfreq      .set    0000033H      ; Volt per frequency ratio
stopfreq      .set    0000034H      ; Target frequecy
stopvolt      .set    0000035H      ; Target voltage
tms_tboost    .set    000003aH      ;
tms_acscale   .set    000003bH      ; convert current
tms_dcscale   .set    000003cH      ; convert voltage
tms_outputb   .set    000003eH      ;
tms_ilmin     .set    000003fH      ;
tblsize       .set    000001aH      ; Setup table size
*
*
ct_swfreg     .word   0804000H      ; Switching frequency timer
ct_port       .word   0804100H      ; Timer control register
ct_phasea     .word   0804200H      ; Phase A timer
ct_phaseb     .word   0804300H      ; Phase B timer
ct_phasec     .word   0804400H      ; Phase C timer
d_output      .word   0804500H      ; General purpurse digital output port
inputcs       .word   0804900H      ; Input voltage and current ADC
acs           .word   0804a00H      ; Phase a output voltage and current ADC
bcs           .word   0804b00H      ; Phase b output voltage and current ADC
ccs           .word   0804c00H      ; Phase c output voltage and current ADC
*
cmd_ad        .int    startcmd      ;
mode_ad       .int    mode_cmd      ;
*
mask_int0     .set    0000001H      ; Set external interrupt 0      (isr0)
mask_int1     .set    0000002H      ; Set external interrupt 1      (isr1)
mask_int2     .set    0000004H      ; Set external interrupt 2      (isr2)
mask_int3     .set    0000008H      ; Set external interrupt 3      (isr3)
mask_timer0   .set    0000100H      ; Set internal timer 0 interrupt

*
*
clear_main    .word   0004444H      ;
reset_out     .word   000ffffH      ;
* Define constants
*
swp_const     .word   10000000
inv11bits     .float  0.00048828125
mil           .float  0.001
```

```
en7             .float   0.0000001
en4             .float   0.0001
AVE             .float   0.2
max             .float   0.95
min             .float   0.05
*
cmd             .usect "dualport",10000h
ctio            .usect "xbus",2000h
lookup          .usect "ram1",400h
varible         .usect "ram2",400h
        .text
init: LDI  0,DP                     ; Point the DP register to page 0
        LDI  00H,ST                 ; Clear and enable cache, and disable OVM (1800h)
        LDI  0000H,IE               ; Clear all interrupts
        LDI  @ctrl,AR0              ; Load peripheral bus memory-mapped reg
        LDI  @xbus,R0               ;
        STI  R0,*+AR0(60H)          ; Init expansion bus control reg
        LDI  @pbus,R0               ;
        STI  R0,*+AR0(64H)          ; Init primary bus control reg
        LDI  @stck,SP               ; Initialize the stack pointer
        CALL init_ct                ; Init counter/timer
        LDI  @d_output,AR0          ;
        LDI  00FFH,R0               ;
        STI  R0,*AR0                ;
        LDI  @ct_port,AR0           ; Pointer for counter/timer control register
        LDI  @reset_out,R0          ;
        STI  R0,*AR0                ; Disable all output
        LDI  @ctrl,AR0              ;
        LDI  @blk1,AR3              ; Sratch pad memory area
        LDI  @dp_mem,AR4            ; Top of dual port memory
        LDI  @blk0,AR5              ; Sratch pad memory area
        LDI  @sram,AR7              ; Top of the look up table
        LDI  @dp_cint,IR0           ; Clear dual port memory interrupt
        LDI  *+AR4(IR0),R0          ;
        LDI  000H,R0                ; Clear sram memory
        RPTS 2047                   ;
        STI  R0,*AR4++(1)           ;
        LDI  @dp_mem,AR4            ; Top of dual port memory
        LDI  0000H,IF               ; Clear all flags
        LDI  0008H,IE               ; Enable interrupt 3 (dual port memory)
        OR   02000H,ST              ; Global interrupt enable
begin:  NOP
            NOP
            NOP
```

```
        NOP
    BR  begin
*
* (NSWC code)
* Initialize counter/timer
*
init_ct: LDI  @ct_port,AR0          ; Pointer for counter/timer control register
    LDI  00ffH,R0                   ;
    STI  R0,*AR0                    ; Disable all counter/timer output
    LDI  @ct_swfreg,AR0             ; Pointer for switching frequency timer 1
    LDI  0034H,R0                   ; Mode 2 (rate generator), 00110100B
    STI  R0,*+AR0(3)                ;
    LDI  0074H,R0                   ; 01110100B
    STI  R0,*+AR0(3)                ;
    LDI  00b4H,R0                   ; 10110100B
    STI  R0,*+AR0(3)                ;
    LDI  @ct_phasea,AR0             ; Pointer for phase a counter
    LDI  0012H,R0           ; Mode 1 (hardware retriggerable one-shoot), 00010010B
    STI  R0,*+AR0(3)                ;
    LDI  0052H,R0                   ; Mode 1, R/W LSB, 01010010B
    STI  R0,*+AR0(3)                ;
    LDI  00b2H,R0                   ; Mode 1, R/W LSB & MSB, 10110010B
    STI  R0,*+AR0(3)                ;
    LDI  @ct_phaseb,AR0             ; Pointer for phase b counter
    LDI  0012H,R0           ; Mode 1 (hardware retriggerable), R/W LSB, 00010010B
    STI  R0,*+AR0(3)                ;
    LDI  0052H,R0                   ; Mode 1, R/W LSB, 01010010B
    STI  R0,*+AR0(3)                ;
    LDI  00b2H,R0                   ; Mode 1, R/W LSB & MSB, 10110010B
    STI  R0,*+AR0(3)                ;
    LDI  @ct_phasec,AR0             ; Pointer for phase c counter
    LDI  0012H,R0           ; Mode 1 (hardware retriggerable), R/W LSB, 00010010B
    STI  R0,*+AR0(3)                ;
    LDI  0052H,R0                   ; Mode 1, R/W LSB, 01010010B
    STI  R0,*+AR0(3)                ;
    LDI  00b2H,R0                   ; Mode 1, R/W LSB & MSB, 10110010B
    STI  R0,*+AR0(3)                ;
    RETS
*(NSWC code)
*Modified 25 Feb 97
read_cmd: LDI  *+AR4(1),R0          ; Check command
    AND  00FFH,R0                   ; Clear all other bits
    CMPI  01EH,R0                   ;
    BHS  stopinit                   ; Ignore command if command >= 23
```

```
          LDI   @cmd_ad,R1            ;
          ADDI  R1,R0                 ;
          BNZ   R0                    ;
stopinit: RETS                        ;
*
startcmd: BR   cmd0                   ; Off
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd10                  ; DC to DC Buck
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          BR   cmd0
          RETS
*
* Turning off ARCP
*(NSWC code)
cmd0:     LDI  08H,IE                 ; Disable interrupts 0,1,2
          LDI  @ct_port,AR0           ; Pointer for counter/timer control register
          LDI  @clear_main,R0         ;
```

```
        STI  R0,*AR0                  ; Disable all output
        STI  R0,*+AR3(tms_outputb)        ;
        LDI  030H,R0                      ;
wait20:  SUBI 01H,R0                      ;
        BNZ  wait20                       ;
        LDI  @reset_out,R0                ;
        STI  R0,*AR0                      ; Disable all counter/timer output
        CALL init_ct
        LDI  00H,R0
        STI  R0,*+AR4(1)
        LDI  @dp_int,IR0
        STI  R0,*+AR4(IR0)
        LDI  @d_output,AR0
        LDI  0FFFH,R0
        STI  R0,*AR0
        LDI  @ct_port,AR0                 ; Pointer for counter/timer control register
        LDI  @reset_out,R0                ;
        STI  R0,*AR0                      ; Disable all output
        RETS                              ;
*
* DC to DC Buck Converter
*(NPS code 5 May 97)
cmd10: LDI  08H,IE                        ; Disable interrupts 0,1,2
        LDI   @ct_port,AR0                ; Pointer for counter/timer control register
        LDI   @clear_main,R0              ;
        STI   R0,*AR0                     ; Disable all output
        LDI   030H,R0                     ;
wait210:  SUBI  01H,R0                    ;
        BNZ   wait210                     ;
        LDI   @reset_out,R0               ;
        STI   R0,*AR0                     ; Disable all counter/timer output
        CALL  init_ct                     ;
        CALL  save_setup                  ; Save data in 32-bit format
        CALL  init_swct                   ; Init switching frequency counters
        LDI   *+AR3(tms_swp),R0
        FLOAT R0                          ;
        LDF   @max,R1                     ;
        MPYF          R0,R1               ;
        STF   R1,*+AR3(UMAX)              ;
        MPYF  @min,R0                     ;
        STF   R0,*+AR3(UMIN)              ;
        LDI   *+AR3(tms_Vref),R0          ;
        FLOAT R0                          ;
        RND  R0                           ;
```

```
        STF   R0,*+AR3(tms_Vref)              ;
*
* start up ramp function
*
        LDI   *+AR3(tms_step),R0              ;
        STI   R0,*+AR3(stopfreq)              ;
        LDI   000H,R0                         ;
        STI   R0,*+AR3(tms_stepx)             ; Startup
        LDI   *+AR3(stopfreq),R0              ;
        FLOAT R0                              ;
        CALL  FPINV                           ;
        MPYF  *+AR3(tms_Vref),R0              ;
        RND   R0                              ;
        STF   R0,*+AR3(vperfreq)              ;
        LDF   0000,R0                         ;
        STF   R0,*+AR3(tms_Vref)              ;
        LDI   @ctrl,AR0                       ; Load peripheral bus memory-
mapped reg
        LDI   *+AR3(tms_delay),R0             ;
        MPYI  064H,R0                         ;
        STI   R0,*+AR0(28H)                   ;
        LDI   @tim0ctl,R0                     ;
        STI   R0,*+AR0(20H)                   ; Init internal timer 0
*
* voltage and current scaling
*
        LDI   *+AR3(tms_dcs),R0               ;
        FLOAT       R0                        ;
        MPYF        @inv11bits,R0             ;
        RND   R0                              ;
        STF   R0,*+AR3(tms_dcscale)           ;
        LDI   *+AR3(tms_acs),R0               ;
        FLOAT       R0                        ;
        MPYF        @inv11bits,R0             ;
        RND   R0                              ;
        STF   R0,*+AR3(tms_acscale)
*
* define hi, hn, hv and T/2
*
        LDI         *+AR3(tms_swf),R0         ; R0 = fsw
        FLOAT       R0                        ;
        MPYF        2.0,R0                    ; R0 = 2*fsw =2*fsamp
        CALL        FPINV                     ; R0 = 1/R0
        LDI         *+AR3(tms_kc),R1          ;
```

```
        FLOAT       R1                          ;
        MPYF @en4,R1                            ; hn
        MPYF R1,R0                              ;
        RND         R0                          ;
        STF         R0,*+AR3(hn)                ; hn*T/2
        LDI         *+AR3(tms_kcb),R0
        FLOAT       R0
        MPYF @en7,R0
        RND         R0
        STF         R0,*+AR3(hv)                ; hv
        LDI         *+AR3(tms_bt),R0
        FLOAT       R0
        MPYF @en4,R0
        RND         R0
        STF         R0,*+AR3(hi)                ; hi
        LDF         0.0,R0                      ;
        STF         R0,*+AR3(Vdiffa)            ; Initialize Vdiff
        STF         R0,*+AR3(Vd_inta)           ; Initialize Vd_int
        STF         R0,*+AR3(Vdiffb)            ; Initialize Vdiff
        STF         R0,*+AR3(Vd_intb)           ; Initialize Vd_int
        LDI   @ct_port,AR0              ; Pointer for counter/timer control register
         LDI   00300H,R0                ; 1100000 (disable phase C)
         STI   R0,*AR0                          ; Enable all counter/timer output
         STI   R0,*+AR3(tms_outputb)
         LDI   010bH,IE                         ; Enable interrupts 0,1,3,8
         LDI   01H,R0                           ;
         STI   R0,*+AR4(1)                      ;
         LDI   @dp_int,IR0                      ;
         STI   R0,*+AR4(IR0)                    ;
         RETS                                   ;
*(NSWC code) Modified 5 May 97
save_setup: LDI tblsize,RC                      ; Init loop counter
         RPTB  save_dp                          ;
         LDI   *AR4++(1),R0             ; Start at the top of the dual port memory
         AND   0ffH,R0                          ; Mask out all higher bits
         LSH   08H,R0                           ; Rotate 8 bits to the left
         LDI   *AR4++(1),R1                     ; Get LSB
         AND   0ffH,R1                          ;
         OR    R0,R1                            ;
save_dp:  STI  R1,*AR3++(1)                     ; Save 32-bit data in internal RAM
         LDI   @dp_mem,AR4                      ; Reset AR4
         LDI   @blk1,AR3                        ; Reset AR3
         LDI   *+AR3(tms_swf),R1                ;
         LDI   @swp_const,R0                    ; Determine switching period
```

```
        CALL  divi                      ;
        STI   R0,*+AR3(tms_swp)         ;
        LDI   02H,R1                    ;
        CALL  divi                      ;
        ADDI  10H,R0                    ;
        STI   R0,*+AR3(tms_swp_120)
        LDI   *+AR3(tms_swp),R0         ;
        LDI   R0,R1                     ; Determine ta
        LSH   -1H,R0                    ;
        STI   R0,*+AR3(tms_ta)
        LDI   *+AR3(tms_dt),R0          ; Determine tb
        LSH   01H,R0                    ;
        SUBI  R0,R1
        LSH   -1H,R1                    ;
        FLOAT R1                        ;
        RND   R1                        ;
        STF   R1,*+AR3(tms_tb)          ;
        LDI   *+AR3(tms_of),R0          ; Determine stepx
        LDI   *+AR3(tms_blk),R1
        MPYI  R1,R0
        LDI   *+AR3(tms_swf),R1
        CALL  divi
        STI   R0,*+AR3(tms_stepx);
        LDI   *+AR3(tms_btime),R1
        STI   R1,*+AR3(tms_tboost)
        LDI   *+AR3(tms_oaci),R2 ;
        FLOAT R2
        STF   R2,*+AR3(tms_ilmin)
        RETS
*(NSWC code)
init_swct:LDI  @ct_swfreg,AR0      ; Pointer for switching frequency timer 1
        LDI  *+AR3(tms_swp),R0     ;
        STI  R0,*+AR0(0)           ; Store LSB of counter 0
        LSH  -08H,R0                       ;
        STI  R0,*+AR0(0)           ; Store MSB of counter 0
        NOP
        NOP
        NOP
        LDI  *+AR3(tms_swp_120),R2 ;
checkout0:LDI  0000H,R0            ;
        STI  R0,*+AR0(3)           ; Latch command
        LDI  *+AR0(0),R0           ;
        AND  000FFH,R0             ; Clear all other higher bits
        LDI  *+AR0(0),R1           ;
```

```
        LSH  0008H,R1
        AND  00f00H,R1                ;
        OR   R1,R0                    ;
        CMPI R2,R0                    ;
        BGT  checkout0                ;
        LDI  *+AR3(tms_swp),R0  ;
        STI  R0,*+AR0(1)              ; Store LSB of counter 1
        LSH  -0008H,R0                ;
        STI  R0,*+AR0(1)              ; Store MSB of counter 1
        NOP
        NOP
        NOP
        LDI  *+AR3(tms_swp_120),R2;
checkout1:LDI  0040H,R0       ;
        STI  R0,*+AR0(3)              ; Latch command
        LDI  *+AR0(1),R0              ;
        AND  000FFH,R0                ; Clear all other higher bits
        LDI  *+AR0(1),R1              ;
        LSH  0008H,R1                         ;
        AND  00f00H,R1                ;
        OR   R1,R0                    ;
        CMPI R2,R0                    ;
        BGT  checkout1                ;
        LDI  *+AR3(tms_swp),R0  ;
        STI  R0,*+AR0(2)              ; Store LSB of counter 2
        LSH  -0008H,R0                ;
        STI  R0,*+AR0(2)              ; Store MSB of counter 2
*
        LDI  @ct_phasea,AR0           ; Pointer for phase a counter
        LDI  *+AR3(tms_btime),R1;
        STI  R1,*+AR0(0)              ; Store LSB of counter 0
        LDI  *+AR3(tms_bdly),R1  ;
        ADDI *+AR3(tms_btime),R1      ;
        STI  R1,*+AR0(1)              ; Store LSB of counter 1
        LDI  *+AR3(tms_ta),R1         ;
        STI  R1,*+AR0(2)              ; Store LSB of counter 2
        LSH  -08H,R1                          ;
        STI  R1,*+AR0(2)              ; Store MSB of counter 2
*
        LDI  @ct_phaseb,AR0           ; Pointer for phase b counter
        LDI  *+AR3(tms_btime),R1;
        STI  R1,*+AR0(0)              ; Store LSB of counter 0
        LDI  *+AR3(tms_bdly),R1  ;
        ADDI *+AR3(tms_btime),R1      ;
```

```
          STI  R1,*+AR0(1)                   ; Store LSB of counter 1
          LDI  *+AR3(tms_ta),R1              ;
          STI  R1,*+AR0(2)                   ; Store LSB of counter 2
          LSH  -08H,R1                               ;
          STI  R1,*+AR0(2)                   ; Store MSB of counter 2
*
          LDI  @ct_phasec,AR0                ; Pointer for phase c counter
          LDI  *+AR3(tms_btime),R1 ;
          STI  R1,*+AR0(0)                   ; Store LSB of counter 0
          LDI  *+AR3(tms_bdly),R1  ;
          ADDI *+AR3(tms_btime),R1           ;
          STI  R1,*+AR0(1)                   ; Store LSB of counter 1
          LDI  *+AR3(tms_ta),R1;
          STI  R1,*+AR0(2)                   ; Store LSB of counter 2
          LSH  -08H,R1                               ;
          STI  R1,*+AR0(2)                   ; Store MSB of counter 2
          LDI   *+AR3(tms_btime),R0          ;
          STI   R0,*+AR3(tms_tboost);
          RETS
*(NSWC code) Modified 25 Feb 97
isr_mode: LDI   *+AR3(tms_mode),R0
          LDI  @mode_ad,R1
          ADDI  R1,R0
          BNZ   R0
          RETS
*
mode_cmd: BR    mode0          ; Stop
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode0
          BR   mode10          ; DC to DC Buck Mode
          BR   mode0
          BR   mode0
*
mode0:           LDI   08H,IE          ; Disable interrupts 0,1,2
          LDI  @ct_port,AR0            ; Pointer for counter/timer control register
          LDI  @clear_main,R0          ;
          STI  R0,*AR0                 ; Disable all output
```

```
        LDI   030H,R0                          ;
wait30:  SUBI  01H,R0               ;
        BNZ   wait30                ;
        LDI   @reset_out,R0         ;
        STI   R0,*AR0               ; Disable all counter/timer output
        AND   08H,IF                ; Clear all pending interrupts 0,1,2
        LDI   00H,R0                ;
        STI   R0,*+AR4(1)           ;
        LDI   @dp_int,IR0           ;
        STI   R0,*+AR4(IR0)         ;
        RETS                  ; Return
*
* DC to DC Buck Converter
*(NPS code) 5 May 97
mode10: LDF *+AR3(tms_Vref),R0                         ;RO=Vref
              LDF           *+AR3(Vin_inv),R1          ;R1= 1/Vin
              LDF           *+AR3(Vout),R2             ;R2= Vout
              LDF           *+AR3(Vdiff),R3            ;*R3= Vdiff(n-1)
              LDF           *+AR3(iL),R6               ;R6 = iL
              SUBF  *+AR3(iout),R6                     ;R6 = iL-iout
              MPYF *+AR3(hi),R6                        ;R6 = hi(iL-iout)
              MPYF3         R0,R1,R4                   ;R4= Dss=Vref/Vin
              SUBF3         R0,R2,R5                   ;R5= Vdiff(n) =Vout-Vref
              ADDF R5,R3                               ;*R3= Vdiff(n)+Vdiff(n-1)
              MPYF *+AR3(hn),R3                        ;*R3= Vd_int=KcT/2
[Vdiff(n)+Vdiff(n-1)]
              ADDF *+AR3(Vd_int),R3     ;*R3= Vd_int(n) = Vd_int + Vd_int(n-1)
              SUBF  R3,R4                              ;R4= D=Dss-Vd_int
              SUBF  R6,R4                      ;R4 = Dss - Vd_int - hi(iL-iout)
              LDF           *+AR3(hv),R6               ;R6 = hv
              MPYF R5,R6                               ;R6 = hv(Vout-Vref)
              SUBF  R6,R4          R4 = Dss - Vd_int - hi(iL-iout) - hv(Vout-Vref)
              LDI           *+AR3(tms_swp),R7 ;
              FLOAT         R7                                ;
              MPYF R7,R4                       ;
              SUBF  R4,R7
              CMPF *+AR3(UMAX),R7              ;
              BLE           LOW                               ;
              LDF           *+AR3(UMAX),R7             ;
 LOW:  CMPF *+AR3(UMIN),R7              ;
              BGT           SAME                       ;
              LDF           *+AR3(UMIN),R7             ;
 SAME:        FIX           R7                                ;
              RETS                             ; Return
```

```
* (NSWC code)
time0:  PUSH R0
        PUSHF       R0
        PUSH  AR0
        LDI         *+AR3(tms_stepx),R0
        ADDI  01H,R0
        STI         R0,*+AR3(tms_stepx)
        CMPI  *+AR3(stopfreq),R0
        BLE         looptimer0
        LDI         000H,R0
        LDI         @ctrl,AR0
        STI         R0,*+AR0(20H)            ; Clear counter
        ANDN mask_timer0,IE                  ; Disable timer interrupt
        POP         AR0
        POPF  R0
        POP         R0
        RETI
looptimer0: LDF *+AR3(vperfreq),R0
        ADDF *+AR3(tms_Vref),R0
        RND         R0
        STF         R0,*+AR3(tms_Vref)
        POP         AR0
        POPF  R0
        POP         R0
        RETI
*
* isr0: Phase a interrupt service rountine
*(NPS code) 5 May 97
isr0:   PUSH   ST                            ; Save registers
        PUSH  IR1
        PUSH  R7
        PUSHF       R7
        PUSH  R6
        PUSHF       R6
        PUSH  R5
        PUSHF       R5
        PUSH  R4
        PUSHF       R4
        PUSH  R3
        PUSHF       R3
        PUSH  R2
        PUSHF       R2
        PUSH  R1
        PUSHF       R1
```

```
                PUSH  R0
                PUSHF           R0
                PUSH  AR0
*               LDI             @bcs,AR0                ; Pointer for DC ADC
                LDI             @ccs,AR2
                LDI             *AR0,R0                 ; initiat conversion
                LDI             *AR2,R1
                LDI             00cH,R2
wait:    SUBI   01H,R2
                BNZ             wait
*
* STORE SAMPLED VOLTAGES AND CURRENTS
*
                LDI             *AR0,R0                 ; READ Vin AND iL
                LDI             *AR2,R2
                LDI             R0,R1
                LSH             04H,R1                  ; R1 = iL
                ASH             -14H,R1
                FLOAT           R1
                LSH             14H,R0                  ; Getting Vin (J7)
                ASH             -14H,R0                 ; R0 = Vin
                FLOAT           R0
                LDI             R2,R3
                LSH             04H,R3                  ; R3 = iout
                ASH             -14H,R3
                FLOAT           R3
                LSH             14H,R2                  ; Getting Vout (J1)
                ASH             -14H,R2                 ; R2 = Vout
                FLOAT           R2
                LDI             009H,R7
wait1:   SUBI   01H,R7
                BNZ             wait1
                LDI             *AR0,R4                 ; READ Vin AND iL
                LDI             *AR2,R6
                LDI             R4,R5
                LSH             04H,R5                  ; R5 = iL
                ASH             -14H,R5
                FLOAT           R5
                LSH             14H,R4                  ; Getting Vin (J7)
                ASH             -14H,R4                 ; R4 = Vin
                FLOAT           R4
                LDI             R6,R7
                LSH             04H,R7                  ; R7 = iout
                ASH             -14H,R7
```

```
            FLOAT      R7
            LSH        14H,R6           ; Getting Vout (J1)
            ASH        -14H,R6          ; R6 = Vout
            FLOAT      R6
            ADDF R4,R0                  ; R0 = Vin  (2)
            ADDF R5,R1                  ; R1 = iL   (2)
            ADDF R6,R2                  ; R2 = Vout (2)
            ADDF R7,R3                  ; R3 = iout (2)
            LDI        009H,R7
wait2:  SUBI 01H,R7
            BNZ        wait2
            LDI        *AR0,R4          ; READ Vin AND iL
            LDI        *AR2,R6
            LDI        R4,R5
            LSH        04H,R5           ; R5 = iL
            ASH        -14H,R5
            FLOAT      R5
            LSH        14H,R4           ; Getting Vin (J7)
            ASH        -14H,R4          ; R4 = Vin
            FLOAT      R4
            LDI        R6,R7
            LSH        04H,R7           ; R7 = iout
            ASH        -14H,R7
            FLOAT      R7
            LSH        14H,R6           ; Getting Vout (J1)
            ASH        -14H,R6          ; R6 = Vout
            FLOAT      R6
            ADDF R4,R0                  ; R0 = Vin  (3)
            ADDF R5,R1                  ; R1 = iL   (3)
            ADDF R6,R2                  ; R2 = Vout (3)
            ADDF R7,R3                  ; R3 = iout (3)
            LDI        009H,R7
wait3:  SUBI 01H,R7
            BNZ        wait3
            LDI        *AR0,R4          ; READ Vin AND iL
            LDI        *AR2,R6
            LDI        R4,R5
            LSH        04H,R5           ; R5 = iL
            ASH        -14H,R5
            FLOAT      R5
            LSH        14H,R4           ; Getting Vin (J7)
            ASH        -14H,R4          ; R4 = Vin
            FLOAT      R4
            LDI        R6,R7
```

```
          LSH       04H,R7                 ; R7 = iout
          ASH       -14H,R7
          FLOAT     R7
          LSH       14H,R6                 ; Getting Vout (J1)
          ASH       -14H,R6                ; R6 = Vout
          FLOAT     R6
          ADDF R4,R0                       ; R0 = Vin  (4)
          ADDF R5,R1                       ; R1 = iL   (4)
          ADDF R6,R2                       ; R2 = Vout (4)
          ADDF R7,R3                       ; R3 = iout (4)
          LDI       009H,R7
wait4:  SUBI  01H,R7
          BNZ       wait4
          LDI       *AR0,R4                ; READ Vin AND iL
          LDI       *AR2,R6
          LDI       R4,R5
          LSH       04H,R5                 ; R5 = iL
          ASH       -14H,R5
          FLOAT     R5
          LSH       14H,R4                 ; Getting Vin (J7)
          ASH       -14H,R4                ; R4 = Vin
          FLOAT     R4
          LDI       R6,R7
          LSH       04H,R7                 ; R7 = iout
          ASH       -14H,R7
          FLOAT     R7
          LSH       14H,R6                 ; Getting Vout (J1)
          ASH       -14H,R6                ; R6 = Vout
          FLOAT     R6
          ADDF R4,R0                       ; R0 = Vin  (5)
          ADDF R5,R1                       ; R1 = iL   (5)
          ADDF R6,R2                       ; R2 = Vout (5)
          ADDF R7,R3                       ; R3 = iout (5)
          MPYF @AVE,R1
          MPYF *+AR3(tms_acscale),R1
          RND       R1
          STF       R1,*+AR3(iL)           ; STORE THE INDUCTOR
CURRENT iL
          MPYF @AVE,R3
          MPYF *+AR3(tms_acscale),R3
          RND       R3
          STF       R3,*+AR3(iout)         ; STORE THE OUTPUT
CURRENT iout
          MPYF @AVE,R2
```

```
        MPYF *+AR3(tms_dcscale),R2
        RND         R2                              ;
        STF         R2,*+AR3(Vout)                  ; STORE Output voltage
Vout
        MPYF @AVE,R0
        MPYF *+AR3(tms_dcscale),R0
        CALL FPINV
        RND         R0
        STF         R0,*+AR3(Vin_inv)               ; STORE 1/Vin
        LDF         *+AR3(Vdiffb),R0
        STF         R0,*+AR3(Vdiff)
        LDF         *+AR3(Vd_intb),R0
        STF         R0,*+AR3(Vd_int)
        CALL isr_mode                   ;
        LDI         @ct_phaseb,AR0                  ; Pointer for phase a counter
        STI         R7,*+AR0(2)                     ; Store LSB of counter 2
        LSH         -08H,R7                         ;
        STI         R7,*+AR0(2)                     ; Store MSB of counter 2
        RND         R5
        STF         R5,*+AR3(Vdiffb)
        RND         R3
        STF         R3,*+AR3(Vd_intb)
        ANDN mask_int0,IF                           ; Clear interrupt 0
*
        POP         AR0
        POPF R0
        POP         R0
        POPF R1
        POP         R1
        POPF R2
        POP         R2
        POPF R3
        POP         R3
        POPF R4
        POP         R4
        POPF R5
        POP         R5
        POPF R6
        POP         R6
        POPF R7
        POP         R7
        POP         IR1
        POP         ST
*       RETI                                        ; Return and enable interrupt
```

```
*
*
* isr1: Phase B interrupt service rountine
*(NPS code) 5 May 97
isr1:   PUSH    ST                              ; Save registers
                PUSH  IR1
                PUSH  R7
                PUSHF       R7
                PUSH  R6
                PUSHF       R6
                PUSH  R5
                PUSHF       R5
                PUSH  R4
                PUSHF       R4
                PUSH  R3
                PUSHF       R3
                PUSH  R2
                PUSHF       R2
                PUSH  R1
                PUSHF       R1
                PUSH  R0
                PUSHF       R0
                PUSH  AR0
*
                LDI         @inputcs,AR0        ; Pointer for DC ADC
                LDI         @acs,AR2
                LDI         *AR0,R0                         ; initiat conversion
                LDI         *AR2,R1
                LDI         00cH,R2
wait5:  SUBI    01H,R2
                BNZ         wait5
*
* STORE SAMPLED VOLTAGES AND CURRENTS
*
                LDI         *AR0,R0                         ; READ Vin AND iL
                LDI         *AR2,R2
                LDI         R0,R1
                LSH         04H,R1                          ; R1 = iL
                ASH         -14H,R1
                FLOAT       R1
                LSH         14H,R0                          ; Getting Vin (J7)
                ASH         -14H,R0                         ; R0 = Vin
                FLOAT       R0
                LDI         R2,R3
```

```
        LSH     04H,R3                          ; R3 = iout
        ASH     -14H,R3
        FLOAT   R3
        LSH     14H,R2                          ; Getting Vout (J1)
        ASH     -14H,R2                         ; R2 = Vout
        FLOAT   R2
        LDI     009H,R7
wait6:  SUBI    01H,R7
        BNZ     wait6
        LDI     *AR0,R4                         ; READ Vin AND iL
        LDI     *AR2,R6
        LDI     R4,R5
        LSH     04H,R5                          ; R5 = iL
        ASH     -14H,R5
        FLOAT   R5
        LSH     14H,R4                          ; Getting Vin (J7)
        ASH     -14H,R4                         ; R4 = Vin
        FLOAT   R4
        LDI     R6,R7
        LSH     04H,R7                          ; R7 = iout
        ASH     -14H,R7
        FLOAT   R7
        LSH     14H,R6                          ; Getting Vout (J1)
        ASH     -14H,R6                         ; R6 = Vout
        FLOAT   R6
        ADDF R4,R0                      ; R0 = Vin  (2)
        ADDF R5,R1                      ; R1 = iL   (2)
        ADDF R6,R2                      ; R2 = Vout (2)
        ADDF R7,R3                      ; R3 = iout (2)
        LDI     009H,R7
wait7:  SUBI    01H,R7
        BNZ     wait7
        LDI     *AR0,R4                         ; READ Vin AND iL
        LDI     *AR2,R6
        LDI     R4,R5
        LSH     04H,R5                          ; R5 = iL
        ASH     -14H,R5
        FLOAT   R5
        LSH     14H,R4                          ; Getting Vin (J7)
        ASH     -14H,R4                         ; R4 = Vin
        FLOAT   R4
        LDI     R6,R7
        LSH     04H,R7                          ; R7 = iout
        ASH     -14H,R7
```

```
            FLOAT       R7
            LSH         14H,R6                          ; Getting Vout (J1)
            ASH         -14H,R6                         ; R6 = Vout
            FLOAT       R6
            ADDF R4,R0                  ; R0 = Vin  (3)
            ADDF R5,R1                  ; R1 = iL   (3)
            ADDF R6,R2                  ; R2 = Vout (3)
            ADDF R7,R3                  ; R3 = iout (3)
            LDI         009H,R7
wait8:  SUBI 01H,R7
            BNZ         wait8
            LDI         *AR0,R4                         ; READ Vin AND iL
            LDI         *AR2,R6
            LDI         R4,R5
            LSH         04H,R5                          ; R5 = iL
            ASH         -14H,R5
            FLOAT       R5
            LSH         14H,R4                          ; Getting Vin (J7)
            ASH         -14H,R4                         ; R4 = Vin
            FLOAT       R4
            LDI         R6,R7
            LSH         04H,R7                          ; R7 = iout
            ASH         -14H,R7
            FLOAT       R7
            LSH         14H,R6                          ; Getting Vout (J1)
            ASH         -14H,R6                         ; R6 = Vout
            FLOAT       R6
            ADDF R4,R0                  ; R0 = Vin  (4)
            ADDF R5,R1                  ; R1 = iL   (4)
            ADDF R6,R2                  ; R2 = Vout (4)
            ADDF R7,R3                  ; R3 = iout (4)
            LDI         009H,R7
wait9:  SUBI 01H,R7
            BNZ         wait9
            LDI         *AR0,R4                         ; READ Vin AND iL
            LDI         *AR2,R6
            LDI         R4,R5
            LSH         04H,R5                          ; R5 = iL
            ASH         -14H,R5
            FLOAT       R5
            LSH         14H,R4                          ; Getting Vin (J7)
            ASH         -14H,R4                         ; R4 = Vin
            FLOAT       R4
            LDI         R6,R7
```

```
            LSH         04H,R7                                    ; R7 = iout
            ASH         -14H,R7
            FLOAT       R7
            LSH         14H,R6                                    ; Getting Vout (J1)
            ASH         -14H,R6                                   ; R6 = Vout
            FLOAT       R6
            ADDF R4,R0                        ; R0 = Vin  (5)
            ADDF R5,R1                        ; R1 = iL   (5)
            ADDF R6,R2                        ; R2 = Vout (5)
            ADDF R7,R3                        ; R3 = iout (5)
            MPYF @AVE,R1
            MPYF *+AR3(tms_acscale),R1
            RND         R1
            STF         R1,*+AR3(iL)          ; STORE THE INDUCTOR
CURRENT iL
            MPYF @AVE,R3
            MPYF *+AR3(tms_acscale),R3
            RND         R3
            STF         R3,*+AR3(iout)                 ; STORE THE OUTPUT
CURRENT iout
            MPYF @AVE,R2
            MPYF *+AR3(tms_dcscale),R2
            RND         R2                                        ;
            STF         R2,*+AR3(Vout)                 ; STORE Output voltage
Vout
            MPYF @AVE,R0
            MPYF *+AR3(tms_dcscale),R0
            CALL FPINV                        ;
            RND         R0                                        ;
            STF         R0,*+AR3(Vin_inv)     ; STORE 1/Vin
            LDF         *+AR3(Vdiffa),R0
            STF         R0,*+AR3(Vdiff)
            LDF         *+AR3(Vd_inta),R0
            STF         R0,*+AR3(Vd_int)
            CALL isr_mode                              ;
            LDI         @ct_phasea,AR0                 ; Pointer for phase a counter
            STI         R7,*+AR0(2)                    ; Store LSB of counter 2
            LSH         -08H,R7                                   ;
            STI         R7,*+AR0(2)                    ; Store MSB of counter 2
            RND         R5
            STF         R5,*+AR3(Vdiffa)
            RND         R3
            STF         R3,*+AR3(Vd_inta)
            ANDN mask_int1,IF             ; Clear interrupt 1
```

```
*
        POP     AR0
        POPF  R0
        POP     R0
        POPF  R1
        POP     R1
        POPF  R2
        POP     R2
        POPF  R3
        POP     R3
        POPF  R4
        POP     R4
        POPF  R5
        POP     R5
        POPF  R6
        POP     R6
        POPF  R7
        POP     R7
        POP     IR1
        POP     ST
*
        RETI                            ; Return and enable interrupt
*
*
*(NSWC code)
* irs3: Dual port memory interrupt service rountine
*
isr3:   PUSH    ST                      ; Save registers
        PUSH DP
        PUSH IR1
        PUSH R7
        PUSHF       R7
        PUSH R6
        PUSHF       R6
        PUSH R5
        PUSHF       R5
        PUSH R4
        PUSHF       R4
        PUSH R3
        PUSHF       R3
        PUSH R2
        PUSHF       R2
        PUSH R1
        PUSHF       R1
```

```
        PUSH  R0
        PUSHF       R0
*
        LDI         @dp_cint,IR0        ;
        LDI         *+AR4(IR0),R0              ; Clear interrupt
        CALL  read_cmd                  ;
        ANDN mask_int3,IF          ; Clear interrupt 3
*
        POPF  R0
        POP         R0
        POPF  R1
        POP         R1
        POPF  R2
        POP         R2
        POPF  R3
        POP         R3
        POPF  R4
        POP         R4
        POPF  R5
        POP         R5
        POPF  R6
        POP         R6
        POPF  R7
        POP         R7
        POP         IR1
        POP         DP
        POP         ST
*
        RETI                                   ; Return and enable interrupt
*
*

    .end
```

LIST OF REFERENCES

1. Dade, T.B., "Advanced Electric Propulsion, Power Generation, and Power Distribution," *Naval Engineers Journal*, Vol. 106, No. 2, pp. 83-92, March, 1994.

2. Blalock, H.C., "Power Electronic Converter Simulation, Real Time Control and Hardware-In-The-Loop Testing for a Shipboard Integrated Power System," Electrical Engineer Thesis, Naval Postgraduate School, Monterey, CA, March, 1995.

3. Office of Naval Research, PEBB Vision Statement, "Introducing 'The Second Electronic Revolution'," http://web.fie.com/htdoc/fed/onr/any/any/text/any/intro.htm, November 1996.

4. Langlois, T.L., The Analysis of Interconnected, High-Power DC-to-DC Converters for DC Zonal Electrical Distribution, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1997.

5. Mohan, N., Undeland, T.M., Robbins, W.P., *Power Electronics, Converters, Applications and Design*, John Wiley & Sons, Inc, New York, 1989.

6. International Rectifier, "IRGTI090U06 Ultra-Fast IGBT Data Sheet," El Segundo, CA, 1996.

7. Badorf, M. G., "Power Electronic Building Block Testbed Stability Criteria and Hardware Validation Studies," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1997.

8. DeDoncker, R.W., Lyons, J.P., "The Auxiliary Resonant Commutated Pole Converter," *IEEE-IAS Annual Meeting Proceedings*, 1990 pp. 1228-1235.

9. Mayer J., Salberta, F., *High-Frequency Power Electronic Converter For Propulsion Applications,* Dept. of EE and Applied Research Laboratory, Pennsylvania State University, University Park, PA.

10. Oberley, M. J., "The Operation and Interaction of the Auxiliary Resonant Commutated Pole Converter in a Shipboard DC Power Distribution Network," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1996.

11. Salerno, B.D., "Controller Design, Analysis, and Prototype for Ship Service Converter Module," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1996.

12. Ifeachor, E.C., Jervis, B.W., *Digital Signal Processing A Practical Approach,* Addison-Wesley Publishers Ltd., 1993.

13. SPECTRUM Signal Processing Inc., *TMS320C30 SYSTEM BOARD User's Manual,* Loughborough, England., August 1990.

14. DSP-CITeco, *LD31/LD31NET User's Guide Version 1.0*, dSPACE digital signal processing and control engineering GmbH, Paderborn, Germany, 1993.

15. NSWC/CDAD Schematics, Code 813, Annapolis, 1995.

16. Texas Instruments, "TMS320C3x User's Guide," Texas Instruments, Inc., 1994.

17. MAXIM, "MAX120/MAX122 Data Sheet," Sunnyvale, CA, 1994.

18. Intel, "MCS - 51 Macro Assembler User's Guide for DOS Systems," Intel Corp.

19. Texas Instruments, "TMS320 Floating-Point DSP Optimizing Compiler," Texas Instruments, Inc., 1991.

20. Texas Instruments, "TMS320 Floating-Point DSP Assembly Language Tools," Texas Instruments, Inc., 1991.

21. Texas Instruments, "TMS320C3X C Source Debugger," Texas Instruments, Inc., 1993.

22. Harris Semiconductor, "82C54 CMOS Programmable Interval Timer Data Sheet," Melbourne, Fl, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.................................2
   8725 John J. Kingman Rd., STE 0944
   Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library..2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3.. Chairman, Code EC...1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. John Ciezki, Code EC/Cy..3
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Robert Ashton, Code EC/Ah...2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

6. Roberto Cristi, Code EC/Cx..1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

7. Roger Cooley...1
   Naval Surface Warfare Center
   Carderock Division/Annapolis Detachment
   3A Leggett Circle
   Annapolis, MD 21402-5067

8. Tuan Duong... 1
   Naval Surface Warfare Center
   Carderock Division/Annapolis Detachment Code 813
   3A Leggett Circle
   Annapolis, MD 21402-5067

7. Ronald J. Hanson.. 2
c/o Earl C. Hanson
10810 Spirit Lake Rd
Grantsburg, WI 54840